Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Vytvoření informačního systému pro inventarizaci na platformě JAVA

J2EE Inventory Information System

2014 Bc. Pavel Podstawka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Pavel Podstawka**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T059 Mobilní technologie

Téma: Vytvoření informačního systému pro inventarizaci na platformě JAVA J2EE Inventory Information System

Zásady pro vypracování:

Cílem práce je vytvořit platformově nezávislý informační systém sloužící k inventarizaci majetku, s jehož využitím bude možné vytvářet a identifikovat položky invenáře pomocí QR kódu.

Cíle práce:
1. Popsat možnosti QR kódu.
2. Návrh a vytvoření databázového schématu sloužícího k ukládání dat.
3. Vytvořit vhodný Informační systém na platformě Java sloužící k invenratizaci položek na základě QR kódu.
4. Implementace vhodného zabezpečeného rozhraní pro přístup aplikace k databázi informačního systému.
5. Otestování systému a vytvoření dokumentace
6. Shrnutí požadovaných výsledků a možnosti dalšího vývoje

Seznam doporučené odborné literatury:

Goncalves, A., Beginning Java EE 7, Apress, 2013, ISBN 978-1430246268.

ZAMBON, Giulio. Beginning JSP, JSF and Tomcat: Java web development. 2nd edition. s.l.: Springer, 2012. ISBN 978-143-0246-237.

HO, Clarence a Rob HARROP. Pro Spring 3. New York: Distributed to the Book trade worldwide by Springer Science+Business Media, c2012, xxx, 912 p. Expert's voice in Spring. ISBN 14-302-4107-1.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Skapa, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 5. května 2014 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2014 .....

Rád bych poděkoval svému vedoucímu práce panu Ing. Janu Skapovi, Ph.D. za jeho vedení. Dále bych chtěl poděkovat Jiřímu Šlachtovi za jeho věcné připomínky a rady. Také bych rád poděkoval Lucii Římovské a své rodině za velkou podporu.

Abstrakt

Cílem této práce je návrh a vytvoření informačního systému založeného pro inventarizaci majetku. Informační systém byl vyvinut na platformě Java EE s pomocí Spring frameworku podporujícího vývoj webových aplikací. Inventarizace majetku je realizována použitím QR kódů, ve kterých jsou uloženy unikátní identifikátory. Vyvinutá aplikace poskytuje možnosti administrace uložených záznamů o zařízeních v databázi. Systém plně nahrazuje předchozí řešení. Na základě zkušeností uživatelů z praxe je také rozšířen o nové funkce.

Klíčová slova: Java, Java EE, Spring, Spring MVC, QR kódy

Abstract

The objective of this thesis is design and implementation of the information system used as the assets inventory. Information system was developed in Java EE with using its Spring framework supporting the web application development. Assets stocktaking is being implemented by QR codes containing unique identifiers. Developed application offers the administration possibilities for the records stored in database. The system fully replaces the previous solution and it has been extended by the new features developed with consideration of the users practical experience.

Keywords: Java, Java EE, Spring, Spring MVC, QR codes

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
ACL	–	Access control list
DFD	–	Data Flow Diagram
DTO	–	Data Transfer Object
EER	–	Enhanced entity–relationship model
EJB	–	Enterprise Java Beans
HTTP	–	Hypertext Transfer Protocol
HTTPS	–	Hypertext Transfer Protocol Secure
IoC	–	Inversion of Control
IS	–	Informační systém
Java EE	–	Java Enterprise Edition
Java SE	–	Java Standard Edition
JDBC	–	Java Database Connectivity
JDK	–	Java Development Kit
JPA	–	Java Persistence API
JPQL	–	Java Presistance Query Language
JSP	–	Java Server Pages
JSON	–	JavaScript Object Notation
JSTL	–	JavaServer Pages Standard Tag Library
JVM	–	Java Virtual Machine
LDAP	–	Lightweight Directory Access Protocol
MVC	–	Model View Controller
ORM	–	Object-relational mapping
PDF	–	Portable Document Format
QR	–	Quick Response
REST	–	Representational State Transfer
SQL	–	Structured Query Language
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locator
WAR	–	Web application ARchive
XHTML	–	Extensible HyperText Markup Language
XML	–	Extensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

Úvod

V prostorách Katedry telekomunikační techniky Vysoké školy báňské - Technické univerzity Ostrava je řada zařízení, která jsou vypůjčována do jiných místností. Tato zařízení jsou označená čárovými kódy. Počet zařízení, která jsou schopná přečíst čárové kódy je však omezený. Označením lehce čitelného kódu pro mobilní přístroje dojde ke zvýšení dostupnosti a celkového přehledu o zařízeních. Stále častější je využití mobilních technologií společně s QR kódy, jelikož moderní telefonní přístroje obsahují relativně kvalitní fotoaparát. Generování QR kódů by tedy mohlo být vhodným řešením, jak uchovávat jednoznačný identifikátor o zařízeních. Je tedy nutné vytvořit takový systém, který bude generovat QR kódy a zároveň uchovávat a poskytovat data o zařízeních.

Jak napovídá zadání, systém bude postaven na platformě Java kvůli jejím specifickým vlastnostem. Tato technologie ovšem obsahuje řadu platforem, které je možné při vytváření aplikace využít. Proto je nutné před samotnou implementací detailně zjistit funkční požadavky kladené na systém. Následně zjištěné požadavky zanalyzovat jak funkčně, tak i datově. Funkční analýza bude zaměřena na správu samotných zařízení a dalších specifických funkcí. Datová analýza slouží ke správnému návrhu schématu pro databázový server, který bude sloužit k uchovávání perzistentních dat. Po zpracování analýz bude následovat samotná implementace systému na konkrétní Java platformu. Práce je tedy především praktického rázu, jelikož jejím hlavním cílem je vytvoření samotné aplikace.

Tato aplikace bude také sloužit ke správě zařízení, kde budou oprávněné osoby provádět různé úpravy. Další funkcí aplikace je poskytnout jednotné zabezpečené rozhraní klientským přístrojům sloužícím pro získávání dat pomocí jednoznačného identifikátoru z QR kódů. Všechny funkce systému je nutné před nasazením do produkčního prostředí nejprve otestovat. Další nedílnou součástí této práce je vytvoření dokumentace, která bude sloužit ke správnému pochopení systému jak z uživatelského, tak z technického hlediska.

1 Možnosti Quick Response kódů

Quick Response kód (QR kód) je speciální typ čárového kódu, do kterého se ukládají data. V současné době jsou často využívány u mobilních technologií. Komerční čárové kódy mají průměrnou maximální kapacitu zhruba dvacet znaků. QR kód je schopen zakódovat desetkrát, až několik set krát větší množství informací.[9] Tento typ kódu charakterizují následující vlastností:

- vysoká kapacita na malé ploše,
- malá velikost,
- možnost korekce chyb,
- nezávislost na úhlu snímání.

1.1 Typy Quick Response kódů

- **QR kód Model 1** - originální QR kód. Největší verze tohoto kódu je 14 (73 x 73 modulů), který je schopen uložit až 1167 číslic,
- **QR kód Model 2** - zlepšení modelu 1 s největší verzi 40 (177 x 177 modulů), který je schopen uložit až 7089 číslic. Když se mluví o QR kódu obecně, obvykle se myslí tento typ,
- **Micro QR kód** - tento typ kódu má pouze jeden detekční vzorek. Díky tomu je kód menší. Největší verze tohoto modulu je M4 (17 x 17 modulů), kde je možné uložit až 35 číslic,
- **iQR Code** - kód, který může být generován buď čtvercově, nebo obdélníkově. Maximální verze může být teoreticky 61 (422 x 422 modulů), které mohou uložit okolo 40 000 číslic,
- **SQRC** - QR kód, který má omezenou funkci čtení. Může být použitý pro ukládání privátních dat,
- **LogoQ** - tento QR kód může být obohacen o ilustrace, písmena a loga.

1.2 Korekce chyb

Velká výhoda QR kódů je korekce chyb. To umožňuje obnovit data v případě, že je QR kód znečištěný, nebo poškozený. Celkem jsou k dispozici čtyři úrovně korekce, které si uživatel může vybrat podle provozního prostředí. Obecně platí, že čím lepší korekce chyb, tím je zapotřebí více redundantních dat.

V prostředí kde se QR kód lehce znečistí, je doporučená korekce Q a H. Úroveň L a M slouží pro čistá prostředí. Nejčastěji je vybrána korekce M. Všechny typy korekcí jsou zobrazeny v tabulce 1.1. [9]

Obrázek 1.1: Typy QR kódů.

Tabulka 1.1: Úroveň korekce chyb u QR kódů.

Schopnost korekce chyby QR kódů	
Úroveň L	Přibližně 7%
Úroveň M	Přibližně 15%
Úroveň Q	Přibližně 25%
Úroveň H	Přibližně 30%

1.3 Využití Quick Response kódů v systému

Tato práce navazuje na pilotní diplomovou práci, kde byla provedená důkladná analýza potřeb pro inventární systém. [14] Výsledky analýzy jsou shrnuty do následujících bodů:

- QR kód musí obsahovat inventární číslo a název výchozí místnosti, kam zařízení patří. Informace musí být dostupné v prostorech, kde není možné připojení k síti,
- QR kód musí být co nejmenší kvůli aplikaci na malé předměty,
- pro oddělení inventárního čísla a podčísla se používá středník,
- výsledný řetězec bude uložen do QR kódu v bytech.

Výsledkem předchozí práce bylo navržení řetězce pro uložená data ve formátu:

```
Inventární číslo zařízení; Název místnosti;
```

Vytvořený řetězec je použit i v této práci s rozdílem, že Katedra telekomunikační techniky využívá k inventárnímu číslu podčíslo, které je odděleno „/“. Data tedy budou uložena v textovém řetězci:

```
Inventární číslo zařízení/podčíslo zařízení; Název místnosti;
```

2 Funkční požadavky

Hlavní myšlenkou, proč vytvářet inventární systém je rychlá a snadná identifikace zařízení. Podle čeho a jakým způsobem zařízení identifikovat je vysvětleno v této kapitole. Dále je nutné rozvést funkce, které má systém obsahovat.

Systém slouží k rychlé identifikaci zařízení pomocí QR kódů. Vhodným zařízením se nasnímá QR kód, který bude nalepen na zařízení. V QR kódu bude zakódován jednoznačný identifikátor, který se odešle do inventárního systému. Inventární systém dohledá informace o zařízení a odešle data v požadovaném formátu zpět do zařízení. Tento jednoduchý pohled je zobrazen na obrázku 2.1.

Diplomová práce se zabývá pouze inventárním systémem, nikoliv klientským mobilním zařízením. Systém bude soběstačný i bez mobilního klienta. Funkce systému jsou popsány v kapitole 2.4. Slouží vyučujícím a administrativním pracovníkům, kteří jsou zodpovědní za určité místnosti. Má napomáhat k lepšímu přehledu o aktuálním stavu, který byl doposud nepřehledný.

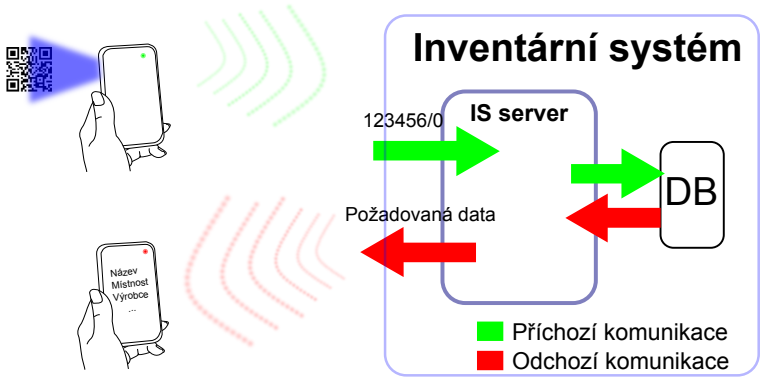
2.1 Vstupy

Systém bude uchovávat data o zařízeních, kdy každé zařízení má svého výrobce a domácí místnost. Do systému také není možné vstoupit bez přihlášení, takže je nutné uchovávat přihlašovací údaje o uživateli. V rámci generování dynamických PDF šablon je nutné uchovávat jejich parametry. Vstupní parametry jsou shrnuty do následujících bodů:

- **zařízení**: název, popis, inventární číslo, podčíslo, sériové číslo, webový odkaz, výchozí místnost, výrobce, aktuální cena, aktuální odpis, běžná účetní hodnota, obrázky k náhledu a souborové přílohy,
- **výrobce**: název, popis, webový odkaz,
- **místnost**: název, popis, správce,
- **uživatel**: uživatelské jméno, uživatelské heslo, jméno, příjmení, e-mailová adresa,
- **parametry šablony**: název, odsazení, velikost, rozestupy, počet QR kódů na řádek.

2.2 Výstupy

Výstupy systému se dělí na hromadné a detailní. Detailní výstupy slouží především k zobrazení podrobných informací o jednom zařízení. Hromadné výstupy obsahují soubor zařízení, kde jsou zobrazeny základní informace. Hromadné výstupy jsou nejčastěji reprezentovány ve formě tabulek.



Obrázek 2.1: Hlavní funkce systému.

- vygenerované QR kódy celé místnosti spolu s inventárním číslem a názvem zařízení,
- emailové zprávy pro zasílání upomínek ke končící lhůtě vypůjčení,
- informace o vypůjčení a historii vypůjčení,
- export dat všech zařízení, které jsou v místnosti,
- export zařízení do PDF souboru.

2.3 Vlastnosti systému

- **Bezpečnost** – do aplikace nebude možné vstoupit bez přihlášení. Je tedy nutné ji zabezpečit tak, aby se neoprávněné osoby do systému nedostaly. Autentizace je nedílnou součástí systému. Dalším důležitým aspektem je autorizace. Každý uživatel nebude mít stejná práva. Různé uživatelské role by měly zajistit, že uživatel bude mít přístup jen do určitých částí systému. Zároveň bude mít uživatel přístup ke svým datům, ale k datům ostatních uživatelů již nikoliv.
- **Flexibilita** – systém by měl být odolný vůči změnám. Neměl by být schopen pracovat pouze na jednom operačním systému. Systém by také neměl být vázaný na jeden konkrétní databázový systém. V případě potřeby by měl umožňovat jeho změnu.
- **Informovanost** – uživatel by měl mít možnost kontaktovat ostatní uživatele v systému, pokud například nevrátili včas zařízení, které si vypůjčili.
- **Dostupnost** – jak již bylo zmíněno, tento systém má zajistit především co největší dostupnost z jakéhokoliv zařízení, které uživatelé budou používat. Aby byla zajištěna komplexní funkcionalita, je nutné vzít v potaz i fakt, že systém bude komunikovat s mobilní aplikací. Ta bude sloužit ke čtení QR kódu a získávání dat z informačního serveru.

2.4 Funkce systému

V předchozí práci bylo možné pouze evidovat zařízení do místností. Toto řešení postrádá smysl z hlediska bezpečnosti, jelikož každý uživatel s rolí správce nemůže editovat zařízení v jiných místnostech, kde není správcem. V systému tedy bude možnost přidělit ke každé místnosti správce, který v ní bude mít veškerá práva. V ostatních místnostech již stejná práva mít nebude. Tím je zaručeno, že si správce nahraje do místnosti jen takové položky, které opravdu chce. Aby byla zajištěna přehlednost v systému, pak funkce přidávání místností připadne pouze administrátorovi, nebo hlavnímu administrátorovi.

Další funkcí je možnost exportu QR kódu a krátkého popisu na lepící štítky. Ty se nalepí na zařízení. Problémem je, že štítků je nepřeberné množství a tak je třeba vymyslet řešení, které bude pokrývat co nejvíce různorodých možností.

Aplikace musí umět hromadné importy a exporty nejlépe ze známých programů jako Microsoft Office, či jeho Open source alternativy. Správci místností by měli mít možnost hromadného smazání zařízení ve svých místnostech. V systému bude možno vytvářet, upravovat a mazat zařízení i jednotlivě.

Historie o vypůjčení zařízení je nutné kvůli rychlému dohledání zařízení v případě ztráty. Zároveň se zabrání tomu, aby si dva uživatelé v jednu chvíli vypůjčili stejné zařízení. Aplikace bude informovat uživatele, že se blíží konec termínu vypůjčení zařízení. Správce místnosti má možnost zaslání rychlé notifikace o neprodleném vrácení zařízení do místnosti.

Každé zařízení lze exportovat do PDF souboru, díky kterému může uživatel informace o zařízení šířit dále.

Funkce lze tedy shrnout do těchto bodů:

- vytvoření, úprava a smazání záznamu o zařízení,
- přidání a odebrání obrázků a příloh zařízení,
- archivace odebraných položek systému,
- přidání, úprava a odebrání výrobce,
- přidání, úprava a odebrání místnosti,
- půjčování zařízení mezi místnostmi,
- nastavení cesty, kde se budou ukládat přílohy,
- možnost základního nastavení přístupu k SMTP serveru,
- šablony pro dynamický tisk QR kódů,
- zasílání emailových zpráv ohledně vypůjčení zařízení.

Hromadné funkce:

- hromadný import/export zařízení,

- hromadné vypůjčení zařízení mezi místnostmi,
- generování QR kódu všech zařízení v místnosti.

2.5 Role uživatelů

V informačním systému jsou role uživatelů rozděleny do tří typů:

- uživatel,
- administrátor,
- hlavní administrátor.

Uživatel, který nemá přidělenou roli je anonymní uživatel a nemá možnost přístupu do systému, pouze mu je zobrazena hlavní stránka, možnost registrace a přihlášení. V systému může být každá role správcem místnosti. Tuto roli nastavuje administrátor, nebo hlavní administrátor. Pokud je role uživatel správcem místnosti, automaticky se mu přidělí pro danou místnost administrátorská práva. V jiné místnosti, kde není správcem, bude mít opět omezené možnosti.
Uživatelská práva:

- zobrazení zařízení, zobrazení místností a výrobců,
- přidání příloh a náhledů k zařízení,
- dočasné vypůjčení zařízení,
- vrácení zařízení (pouze v případě, že si zařízení vypůjčil),
- vytvoření, úprava výrobců,
- smazání výrobce, na kterého nemá žádné zařízení referenci,
- export detailu zařízení do PDF,
- export QR kódů zařízení v místnosti do různých šablon,
- export všech zařízení v místnosti do xsl/xslx a csv formátu,
- zobrazení a editace pouze svého uživatelského profilu.

Administrátorská práva:

- administrátor má všechna uživatelská práva,
- vytváření uživatelů s rolí uživatel,
- vytváření, úprava a mazání místnosti,
- úprava uživatelských účtů (nikoliv aktivace či udělení jiných rolí),

- import zařízení do místnosti z xsl/xslx souborů,
- mazání příloh,
- editace a mazání zařízení,
- možnost vrácení/zrušení rezervace, případně odesílat notifikační e-maily o vrácení zařízení,
- vytváření PDF šablon v systému,
- zobrazení a trvalé odstranění smazaných zařízení.

Práva hlavního administrátora:

- hlavní administrátor má všechna administrátorská práva,
- nastavení detailů emailového klienta,
- nastavení cesty pro ukládání příloh na disk,
- mazání uživatelských účtů,
- nastavení uživatelských rolí,
- povolení, či vyřazení účtů.

2.6 Nefunkční požadavky

Systém by mělo být možné spustit na kterémkoliv operačním systému, zejména na Linuxu. To bylo omezení předchozí práce psané v technologii .NET, jelikož je převážně určena pro Microsoft Windows.

V případě, že bude systém využíván větším počtem uživatelů, je nutné, aby i pod větší zátěží systém rychle reagoval. To znamená, že by hardwarové vybavení serveru nemělo omezovat správný chod IS.

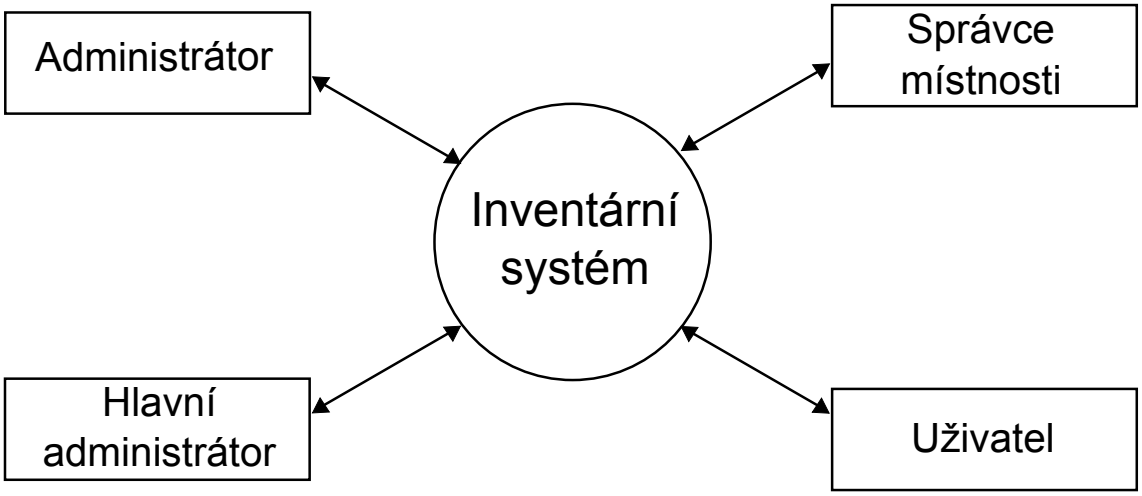
Server by měl komunikovat prostřednictvím protokolu HTTP a HTTPS. Na serveru bude použit reverzní proxy server, který umožňuje přístup jak HTTP, tak HTTPS protokolu. Je čistě na rozhodnutí administrátora serveru, kterou z uvedených metod zvolí v rámci prostředí, jelikož toto zabezpečení je na úrovni serveru, nikoliv aplikace.

2.7 Okolí

Systém bude komunikovat s těmito aktéry:

- Správce místnosti
- Uživatel
- Administrátor
- Hlavní administrátor

Komunikaci znázorňuje kontextový diagram, který je zobrazen na obrázku 2.2.



Obrázek 2.2: Kontextový diagram.

3 Analýza informačního systému

Před implementací je nutné zanalyzovat funkční požadavky systému. To je provedeno datovou a funkční analýzou. Datová analýza je zaměřena na ukládání perzistentních dat. Funkční analýza se zabývá funkcemi systému. Funkcí v systému je ale velké množství a tak se funkční analýza zaměřuje především na administraci zařízení.

3.1 Datová analýza

Při návrhu databáze se musí zohlednit požadavky systému. Požadavky důležité pro vytvoření databáze jsou:

- každé zařízení má výrobce a výchozí místnost,
- v systému může být více zařízení od jednoho výrobce,
- v jedné místnosti může být více zařízení,
- každá místnost má svého správce,
- každé zařízení lze dočasně přesunout do jiné místnosti,
- systém musí umět zobrazit historii přesunů zařízení,
- každé zařízení má status, který označuje polohu zařízení,
- k zařízením je možné přidávat přílohy a obrázkový náhled,
- systém musí být schopen generovat QR kódy do PDF dynamicky – tj. pomocí šablon,
- pro přístup do systému je nutná registrace,
- systém by měl evidovat smazaná zařízení,
- počet evidovaných zařízení na katedře je přibližně 5000.

U většiny tabulek je také informace o datu vytvoření a poslední úpravě záznamu. K těmto datům je navíc přidán login uživatele, který akci provedl. Datová analýza je provedena formou lineárního zápisu a datového slovníku. Grafický návrh je zobrazen pomocí EER diagramu.

3.1.1 Lineární zápis

Lineární zápis je přehledným zápisem databázových tabulek a jejich sloupců. Primární klíče jsou ve tvaru „*primární klíč*“ a cizí klíče ve tvaru „**cizí klíč**“

> **users** {*id*, username, password , enabled, firstName, lastName, email, created, createdByUser, updated, updatedByUser}

authorities {*id*, **userId**, authority}

producers {*id*, title, description, web, created, createdByUser, updated, updatedByUser, deleted}

rooms {*id*, title, description, created, createdByUser, updated, updatedByUser, deleted, **userId**}

devices {*id*, inventoryNumber, inNumber, title, serialNumber, description, iMActual, web, prevailingAccountingValue, actualWrite-off, created, createdByUser, updated, updatedByUser, deleted, **producer**, **homeroom** }

devicefiles {*id*, **deviceId**, filename, description, created, createdByUser, updated, updatedByUser, deleted, absolutePath }

deviceimages {*id*, **deviceId**, filename, description, createdDate, createdByUser, updated, updatedByUser, deleted, absolutepath}

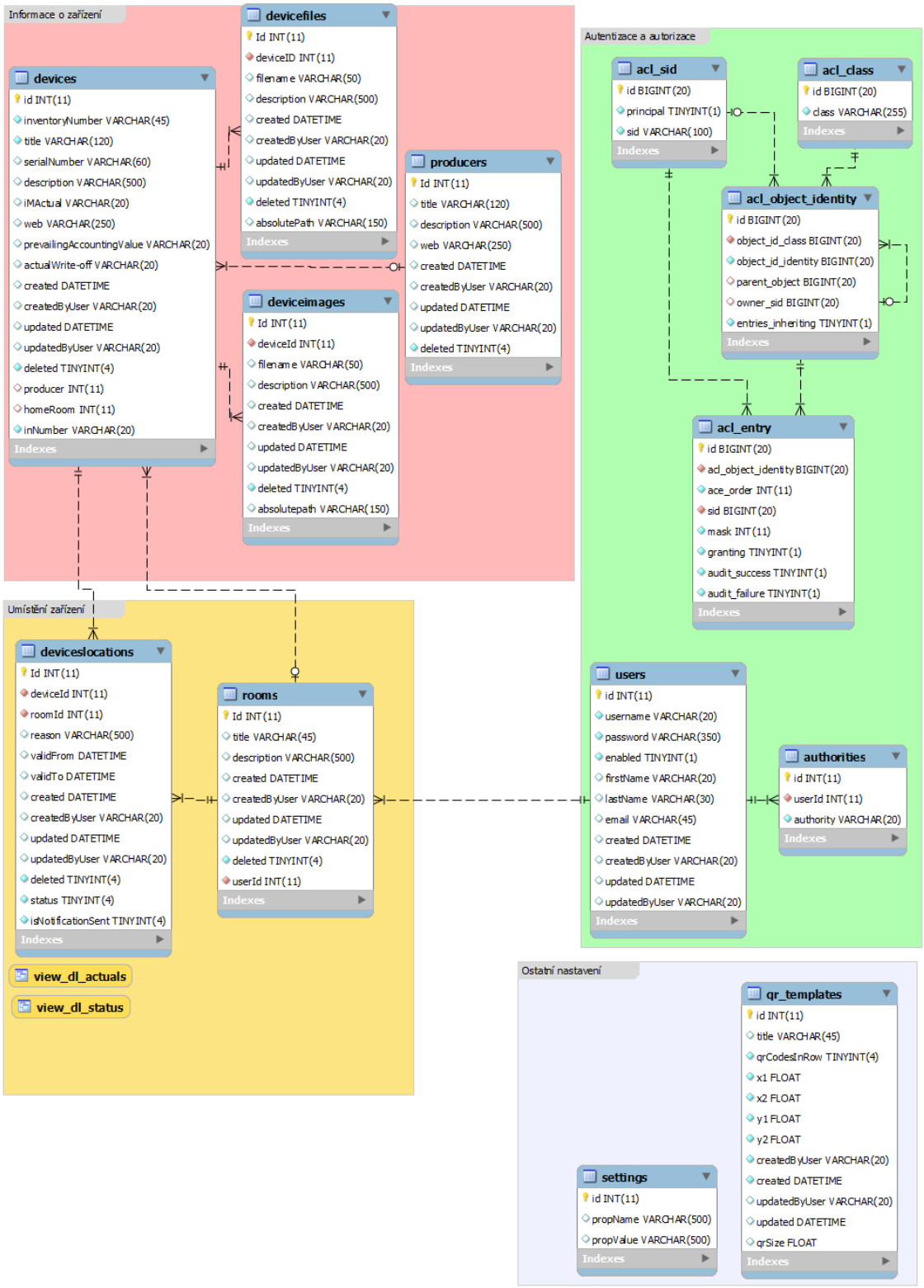
deviceslocations {*id*, **deviceId**, roomId, reason, validFrom, validTo, created, createdByUser, updated, updatedByUser, deleted, status, receivedDate, isNotificationSent, extend, isActual}

qr_templates {*id*, qrSize, title, qrCodesInRow, x1, x2, y1, y2, created ,createdByUser, updated, updatedByUser, textLength}

settings {*id*, propName, propValue}

3.1.2 EER diagram

Pomocí EER diagramu lze vidět propojení jednotlivých tabulek. Návrh je celkově rozdělen do čtyř částí. V části **informace o zařízení** jsou tabulky ohledně příloh a obrázkových náhledů. Také jsou do této sekce zařazeni výrobci. Do části **umístění zařízení** jsou navržené tabulky o názvu místnosti a výpisu přesunů zařízení. Znázorněny jsou zde i pohledy, které jsou popsány níže. Další části je **Autentizace a autorizace**, která slouží pro ověřování přístupu a uchovávání uživatelských dat. Poslední části jsou **ostatní nastavení**, kde je možné najít nastavení systému a také PDF šablony. EER diagram je možné vidět na obrázku 3.1.



Obrázek 3.1: EER diagram.

Datový slovník je možné vidět v příloze na DVD. V databázi jsou použity pohledy, kterými lze porovnávat data vypůjčení zařízení. První pohled *view_dl_status* zobrazuje všechny záznamy z *devicelocations*. Tento pohled má čtyři nové sloupce:

- zda je zařízení rezerováno,
- zda čas vypůjčení vypršel,
- zda schází jen jeden den do vrácení pro zasílání notifikací,
- zda ještě zařízení nebylo vráceno.

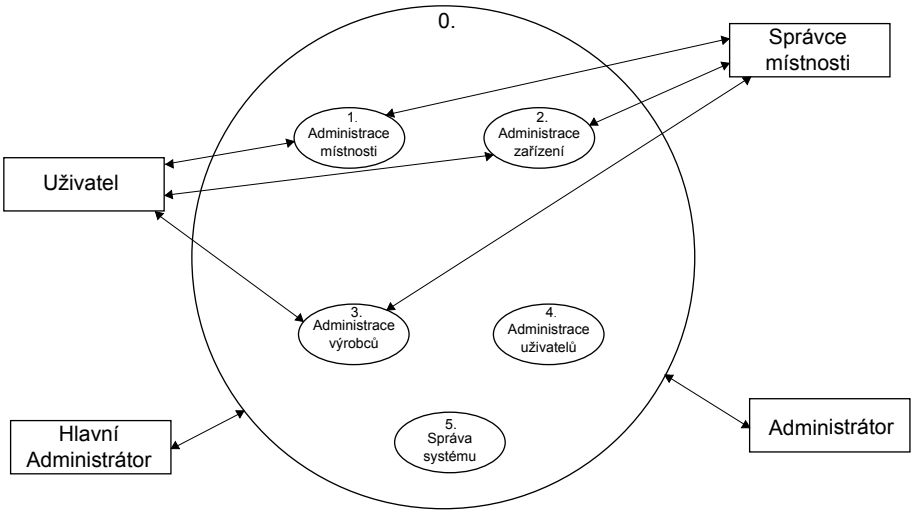
Druhý pohled *view_dl_actuals* je téměř stejný jako pohled *view_dl_status* s tím rozdílem, že zobrazuje aktuální lokaci zařízení. Obsahuje tedy maximálně jeden záznam o každém zařízení. Pohledy napomáhají systému v zobrazení správního statusu zařízení. Práce původně byla navržena pouze pro nastavení uživatelských rolí (tabulky *users* a *authorities*). Role ale nebyly dostačující pro vyvíjející se požadavky systému. To proto, že role uživatel může spravovat určité místnosti a tedy mít v místnosti rozšířenější práva (statut správce). V místnostech, kde nemá rozšířená práva (není tedy správcem) již nikoliv. Proto do systému byly implementovány přístupové listy.

3.2 Funkční analýza

Inventární systém se skládá z řady funkcí, které jsou pro lepší přehled rozepsány do DFD diagramů. Funkce jsou rozděleny na čtyři subsystémy. V této práci je analýza zaměřena především na administraci zařízení, jelikož kvůli zařízením je vytvořen tento IS. Subsystémy jsou:

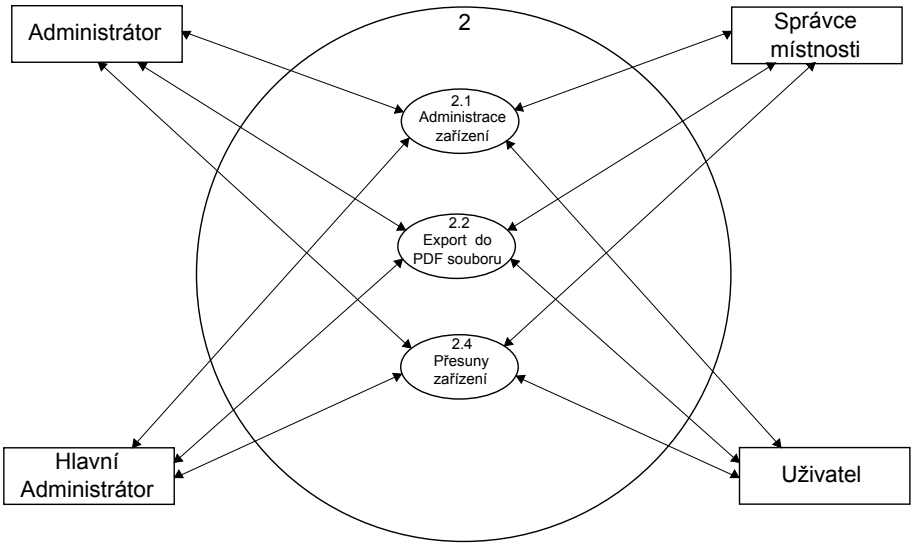
- administrace místností,
- administrace zařízení,
- administrace výrobců,
- administrace nastavení systému.

3.2.1 DFD 0. úrovně



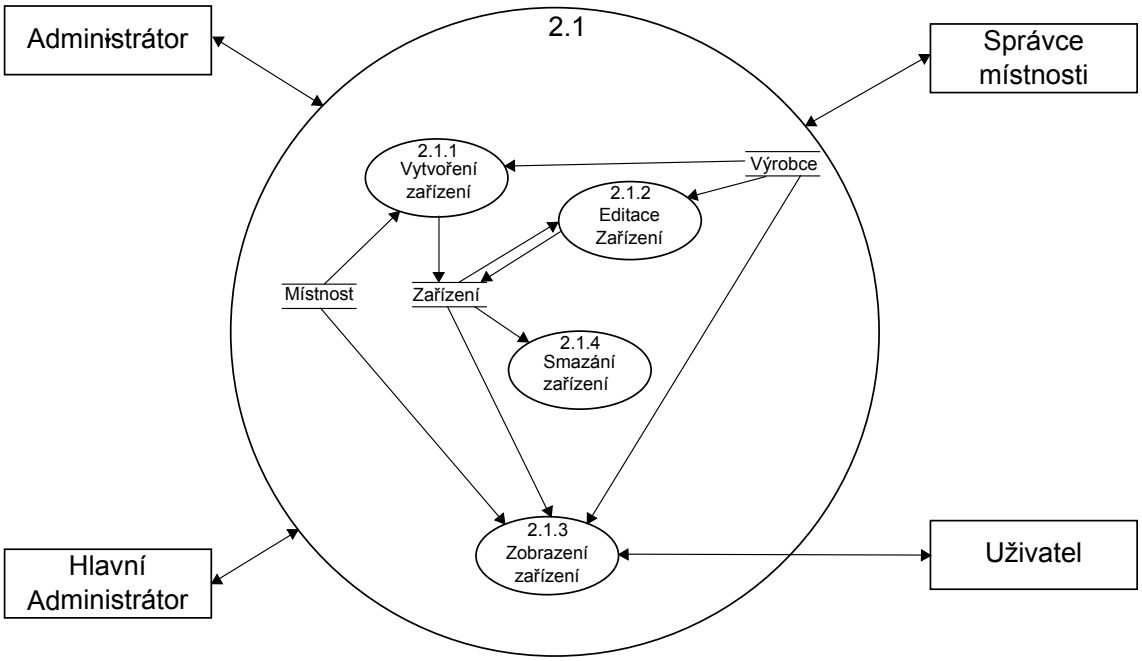
Obrázek 3.2: DFD 0. úrovně.

3.2.2 DFD 1. úrovně - Administrace zařízení



Obrázek 3.3: DFD 1. úrovně.

3.2.3 DFD 2. úrovně - Administrace zařízení



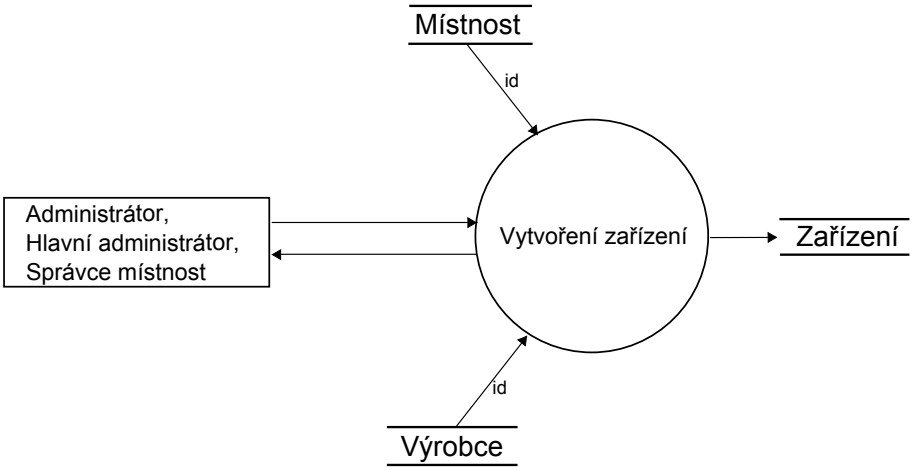
Obrázek 3.4: DFD 2. úrovně Administrace zařízení.

3.3 Minispecifikace

V této sekci jsou popsány minispeficikace zaměřené na zařízení. Každá minispecifikace obsahuje grafické znázornění pomocí DFD diagramu a strukturovaný popis. Strukturovaný popis obsahuje tři různé styly. Kurzívou jsou popsány jednotlivé sloupce databáze.Tučným písmem jsou zvýrazněna formulářová data. Tučným písmem a kurzívou jsou označeny názvy tabulky v databázi. Celkem jsou popsány čtyři minispecifikace:

- vytvoření zařízení,
- úprava zařízení,
- zobrazení zařízení,
- vypůjčení zařízení do jiné místnosti.

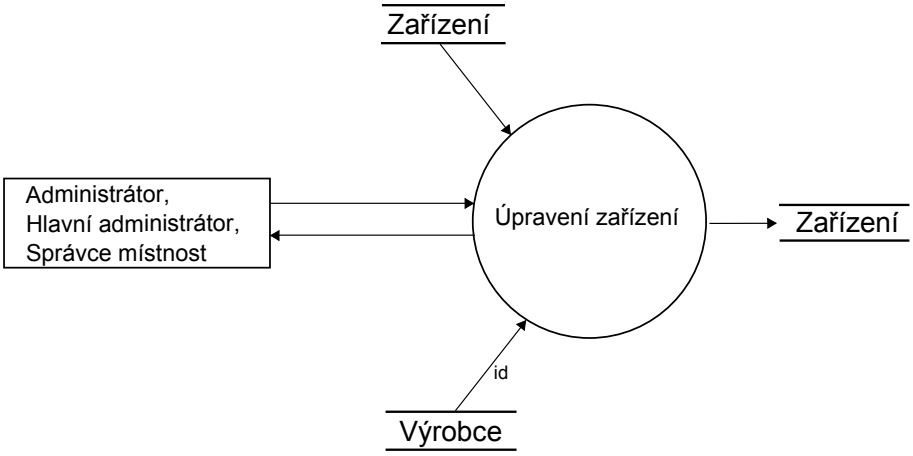
3.3.1 Vytvoření zařízení



Obrázek 3.5: DFD diagram vytvoření zařízení.

Postup vytvoření zařízení (termíny v uvozovkách „*x*“ jsou příkazy, které vykoná systém – jsou psány rozkazovacím způsobem):

1. „Zobraz“ formulář pro přidání zařízení do systému. Do rozbalovací nabídky **Místnosti** „načti“ všechny názvy místností z tabulky ***rooms***. Do rozbalovací nabídky **Výrobce** „načti“ všechny názvy výrobců z tabulky ***producers***.
2. Administrátor, Hlavní administrátor, nebo Správce místnosti vyplní data k zařízení.
3. Po stisknutí tlačítka s názvem **Vytvořit** „zkontroluj“, zda je inventární číslo a podčíslo opravdu číslo, zda je délka názvu alespoň čtyři znaky.
4. Po validaci „zkontroluj“, zda kombinace *inventoryNumber* (inventární číslo) a *inNumber*(podčíslo) již existuje v databázi
5. „Přidej“ k zařízení uživatelské jméno a datum vytvoření záznamu z důvodu bezpečnosti.
6. „Vlož“ záznam o zařízení do tabulky ***devices***.
7. „Zobraz“ formulář pro zadávání údajů s prázdnými údaji a zobraz report o úspěšném vložení zařízení do databáze.
8. Administrátor, Hlavní administrátor, nebo Správce místnosti může pokračovat s prací od bodu 1.



Obrázek 3.6: DFD diagram pro úpravu zařízení.

3.3.2 Úprava zařízení

1. „Zobraz“ formulář pro úpravu zařízení v systému. Do polí „načti“ konkrétní data zařízení z tabulky ***devices***. Do rozbalovací nabídky **Výrobce** „načti“ všechny názvy výrobců z tabulky ***producers***.

2. Administrátor, Hlavní administrátor, nebo Správce místnosti upraví data k zařízení.

3. Po stisknutí tlačítka Upravit „zkontroluj“, zda je inventární číslo a podčíslo opravdu číslo, zda je délka názvu zařízení alespoň čtyři znaky.

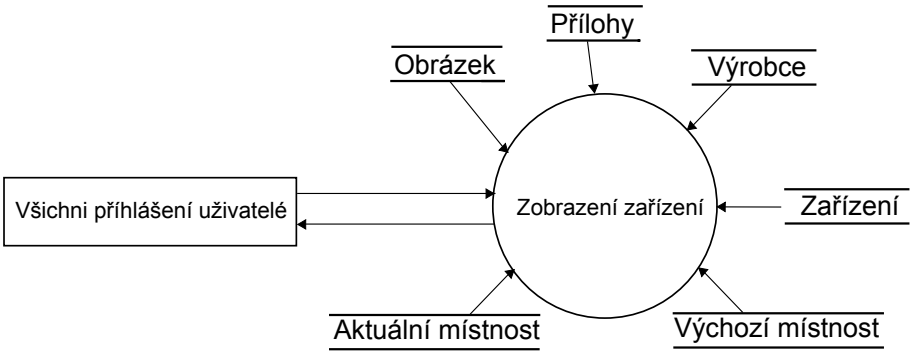
4. „Přidej“ k zařízení uživatelské jméno a datum poslední úpravy záznamu z důvodu bezpečnosti.

5. „Uprav“ záznam o zařízení do tabulky ***devices***.

6. „Přesměruj“ Administrátora, Hlavního administrátora, nebo Správce místnosti na zobrazení místnosti a „zobraz report“ o úspěšném upravení zařízení do databáze.

7. Administrátor, Hlavním administrátor, nebo správce místnosti může pokračovat s prací od bodu 1.

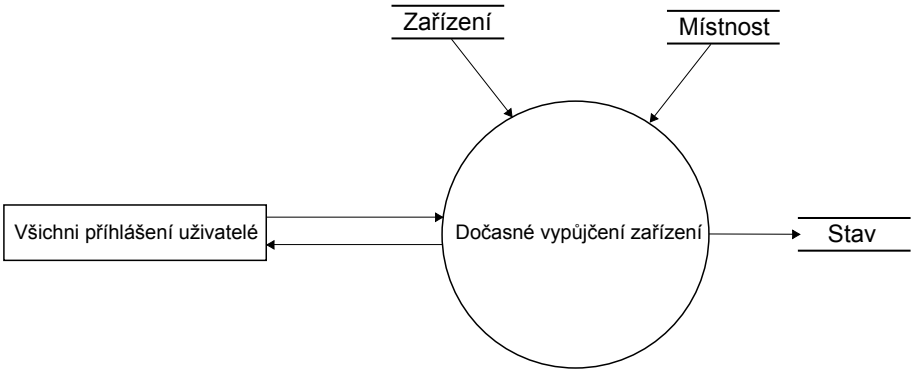
3.3.3 Zobrazení zařízení



Obrázek 3.7: DFD diagram pro zobrazení zařízení.

1. „Zobraz“ tabulku se všemi zařízeními.
2. Uživatel vybere specifické zařízení kliknutím na název zařízení. Do systému se odešle *id* konkrétního zařízení.
3. „Načti“ *id* zařízení a „vyhledej“ data v tabulce ***devices***. V tabulce ***devices*** podle sloupce *producer* „vyhledej“ název výrobce z tabulky ***producer***. V tabulce ***devices*** podle sloupce *homeRoom* „vyhledej“ název výchozí místnosti v tabulce ***rooms***.
4. „Porovnej“, zda pohled ***view_dl_actual*** v sloupci *deviceId* obsahuje *id* zařízení. Pokud ano, „vyhledej“ v pohledu ***view_dl_actual*** pomocí sloupce *roomId* aktuální místnosti z tabulky ***rooms***. Pokud ne, „přeskoč“ tento krok.
5. „Porovnej“, zda v tabulce **deviceimages** sloupec *deviceId* obsahuje *id* zařízení. Pokud ano, „zobraz“ obrázek. Pokud ne, „přeskoč“ tento krok.
6. „Porovnej“, zda v tabulce ***devicefiles*** sloupec *deviceId* obsahuje *id* zařízení. Pokud ano, „zobraz“ všechny přílohy s tímto *id*. Pokud ne, „přeskoč“ tento krok.
7. „Zobraz“ uživateli všechna vyhledaná data

3.3.4 Vypůjčení zařízení do jiné místnosti



Obrázek 3.8: DFD diagram pro vypůjčení zařízení.

1. „Zobraz“ formulář pro vypůjčení zařízení. Do sloupce místnosti „načti“ všechny názvy místnosti z tabulky ***rooms***.
2. Uživatel vybere název místnosti a zadá do formuláře **datum od**, **datum do** a **důvod přesunu**.
3. „Zkontroluj“ datum od a datum do, které uživatel zadal.
4. „Načti“ všechny řádky v tabulce ***deviceslocations***, které ve sloupci *deviceId* obsahují *id* zařízení. Pokud není načten žádný řádek, „přejdi“ ke kroku číslo 6.
5. Pro každý řádek „proveď“.
   5.1. „Zkontroluj“ sloupec *validFrom* a *validTo* z tabulky ***deviceslocations*** s datem zadaným ve formuláři. Pokud se data nepřekrývají, přejdi na bod 6. Pokud se data překrývají, „vypiš“ chybovou hlášku a „přejdi“ k bodu 1.
6. „Zapiš“ do databáze ***deviceslocations*** nový záznam.
7. „Vypiš“ oznámení o úspěšné rezervaci a „přejdi“ na bod číslo 1.

3.4 Komunikace s uživateli

Komunikace s uživateli bude probíhat pomocí webového rozhraní. To je vhodné navrhnout takovým způsobem, aby bylo co nejvíce uživatelsky přívětivé. To znamená, že by se vzhled stránky měl podobat klasickým internetovým stránkám. Díky tomu se uživatel bude v systému lehce orientovat a zároveň se zvýší efektivita jeho práce. V teto kapitole jsou popsány základní prvky, které bude webové rozhraní obsahovat.

3.4.1 Menu lišta

Uživateli bude po přihlášení do systému zobrazena *Menu lišta*, která se bude nacházet na vrcholu webové stránky. Po kliknutí na menu položku zobrazí podmenu, které bude zobrazovat další akce. Menu lišta bude obsahovat celkem 6 položek, které jsou popsány níže:

- **inventární systém** - stisknutím tlačítka *Inventární systém* bude uživateli zobrazen rychlý přehled,
- **zařízení** - v této sekci bude možné přidat zařízení, zobrazit všechna zařízení a administrátoři mohou zobrazit smazané položky,
- **výrobce** - v této liště bude možné vytvořit výrobce a zobrazit všechny výrobce,
- **místnosti** - při otevření položky budou moci administrátoři přidat zařízení a všichni uživatelé zobrazit místnosti,
- **účet** - zde si bude moct každý uživatel zobrazit a upravit svůj uživatelský profil. V této sekci je možné odhlášení ze systému,
- **správa webu** - tato položka je zobrazena pouze administrátorům a obsahuje nastavení systému.

3.4.2 Formuláře

Formuláře budou tvořeny jednotným způsobem a jsou složeny z těchto prvků:

- textové pole,
- rozbalovací nabídka,
- tlačítko,
- kalendář.

Formulář bude zobrazen při těchto akcích:

- přidávání zařízení/místností/výrobců,
- editace zařízení/místností/výrobců,
- filtry pro zobrazení všech zařízení/místností/výrobců,
- přesuny zařízení,
- vytvoření/registrace uživatele,
- nastavení systému,
- vytvoření/editace PDF šablony.

3.4.3 Zpětná vazba

Po vykonání určité akce bude uživateli zobrazená zpětná vazba. Obecně lze shrnout do dvou případů.

- **Úspěšná zpětná vazba** – tato hláška se uživateli zobrazí v případě, že úspěšně provedl akci. Příklad úspěšné zpětné vazby je odeslání potvrzení o úspěšném vytvoření zařízení.
- **Neúspěšná zpětná vazba** – je zobrazena v případě neúspěšné akce. Systém často napovídá, jakou akci musí uživatel provést, aby byla akce úspěšná. Chybová hláška se zobrazí například tehdy, chce-li uživatel vložit zařízení do místnosti, ve které není správcem.

3.4.4 Hromadné výpisy

Hromadné výpisy budou sloužit pro zobrazení velkého množství informací. Výpisy budou uspořádány do tabulkových tvarů. V systému budou hromadné výpisy použity pro tyto akce:

- zobrazení všech zařízení,výrobců a místností,
- přehled zobrazení uživatelů,
- výpis vytvořených PDF šablon,
- zobrazení historie přesunu.

4 Implementace

Tato kapitola je rozdělená do několika částí. První část je věnována MySQL databázi, která byla vybrána pro komunikaci s informačním systémem. Další část je věnována problematice volby architektury, kde jsou popsány výhody a nevýhody jednotlivých řešení. Na architekturu navazuje stručný popis Java EE platformy, která byla vybrána pro chod IS. Praktická část diplomové práce využívá Spring Framework, který je vysvětlen v další kapitole. Další kapitolou je popis implementace. Jsou zde popsány praktické ukázky a způsob řešení určitých funkcí.

4.1 MySQL databáze

V každém větším systému jako je tento je nutné uchovávat perzistentní data, kde budou uloženy údaje o uživatelích, zařízeních, nastavení a podobně. K tomu slouží databáze, které jsou vyvíjeny právě pro tuto potřebu. V této době je velké množství dostupných databází, které se od sebe liší syntaxi a způsobem ukládání dat. Databáze, která byla vybrána pro tento systém je MySQL pro její výhody:

- **stabilita** - každá nová verze je vždy důkladně otestována
- **rychlost** - MySQL dokáže zpracovat velké množství požadavků v relativně krátkém čase,
- **podpora** - MySQL databáze je portována pro většinu operačních systémů,
- **cena** - tato databáze standardně zdarma
- **rozšířenost** - výhodou, která plyne z rozšířenosti je velká uživatelská podpora. [15]

Aplikace této práce se neváže pouze na jeden typ databáze. Z velkého množství databází byla vybrána MySQL především pro její rozšířenost. Veškeré skripty potřebné k chodu databáze jsou obsaženy v příloze na DVD. V případě změny databáze je pro funkci IS nutné zachovat stejnou tabulkovou strukturu a změnit v konfiguračním souboru dialekt a přístupové údaje. Tento postup je popsán v příloze na DVD.

4.2 Volba architektury

V této práci byly zváženy dvě nejvýhodnější varianty architektury. První variantou je vytvoření aplikace, která se bude spouštět přímo v operačním systému uživatele. Tato aplikace by komunikovala s databází na sdíleném úložišti. Velkou výhodou by byla bezpečnost v případě, že by se aplikace poskytovala jen oprávněným osobám. Nevýhodou je udržitelnost vývoje aplikace, která by komunikovala s databází. Stačila by pouze změna SQL syntaxe databáze z důvodu aktualizace databázového systému, nebo migrací na jiný systém. V tomto případě by se aplikace musela adaptovat na nové databázové řešení. V případě složitých výběrů a operací by vzrostla náročnost i na mobilní aplikaci, což je nežádoucí.

Druhou možností je nasadit aplikaci rovnou na server. Aplikace na serveru by pak jednoduše komunikovala s databází. Bylo by nutné vytvoření webového rozhraní, ke kterému by se přistupovalo přes webový prohlížeč. Oproti první metodě by se zvýšila dostupnost systému a odpadly by problémy se zprovozněním aplikace v operačním systému, jelikož by stačilo mít pouze webový prohlížeč. V tomto případě by vytvoření komunikace s mobilní aplikací nemělo být tak obtížné, jako tomu je v prvním případě. Použitím jednotného komunikačního protokolu odpadají problémy způsobené vazbou na konkrétní technologie. Nevýhodou je větší riziko napadení, jelikož webové rozhraní by bylo dostupné prakticky odkudkoliv. Po dohodě s vedoucím práce byla zvolena druhá varianta, a tedy nasazení aplikace na server, odkud bude přístupná dalším uživatelům.[5]

Jedním z požadavků vedoucího práce byla možnost nasazení aplikace nejen pro Katedru telekomunikační techniky, ale i do různých oblastí, kde je potřebná inventarizace. To ovšem znemožňuje připojení přes LDAP, který by usnadnil přístup uživatelům z VŠB-TU Ostrava pomocí jednotného jména a hesla.

4.3 Webová aplikace

Během krátké historie se World Wide Web vyvinul ze sítě obsahující zejména statické informace až po sítě poskytující dynamické informace jako například virtuální obchody a komplexní sociální sítě. Webové aplikace spravují tyto služby a jejich využití může být prakticky kdekoliv. Webové aplikace jsou postavené na architektuře klient-server. Webový prohlížeč je tzv. tenkým klientem, jelikož pouze zobrazuje data.

Nejčastěji jsou tyto aplikace postavené na třívrstvé architektuře:

- **Prezentační vrstva** – tato vrstva slouží k zobrazení dat na klientském zařízení. Data jsou zobrazována webovým prohlížečem.
- **Aplikační vrstva** – je prostředníkem mezi prezentační a datovou vrstvou, která běží na serveru. Obsahuje aplikační logiku, která pracuje s daty.
- **Datová vrstva** – obsahuje aplikační data. Tato data jsou většinou uložena v relační databázi. Dále vrstva obsahuje rozhraní, které slouží pro práci s daty.

Výhody webových aplikací jsou shrnuty do následujících bodů:

- **Tenký klient** – tenkým klientem se nejčastěji rozumí webový prohlížeč.
  - Veškeré výpočty se provádí na serveru.
  - Eliminace přímého přístupu k databázi.
- **Dostupnost** – klient, tedy webový prohlížeč je dostupný téměř na každém počítači a mobilním přístroji.
- **Jednoduchá správa aplikací** – odstraňuje problém s aktualizacemi nových verzí na klientské straně.

Webové aplikace mají také své nevýhody. Velkým problémem je doba odezvy při odesílání a přijímání dat po sítí. Tento problém jde z části eliminovat tím, že se specifické operace přenesou na stranu klienta (např. JavaScript). Další možnou nevýhodou je nadměrné vytížení serveru, který vykonává výpočetní operace všech klientů.

K programování webových aplikací existuje velké množství jazyků jako například: Java, Perl, Ruby, ASP, PHP a další. Inventární systém je naprogramovaný v jazyce Java. Tato technologie je popsaná v další kapitole.

4.4 Java

Informační systém je psán v jazyce Java, který představila firma Sun Microsystems v roce 1995. V roce 2010 Oracle odkoupil Sun Microsystems a od té doby jejich inženýři pracují společně na tom, aby vytvořili plně integrovaný systém a optimalizovaná řešení navržená k lepšímu výkonu.

Technologie Java je programovacím jazykem i platformou. Programování v jazyce Java je objektově orientované a má tedy vlastní syntaxi a styl. Platforma Java je specifickým prostředím, ve kterém se spouští aplikace psané v Java jazyce. V této době má Java čtyři platformy. [6]

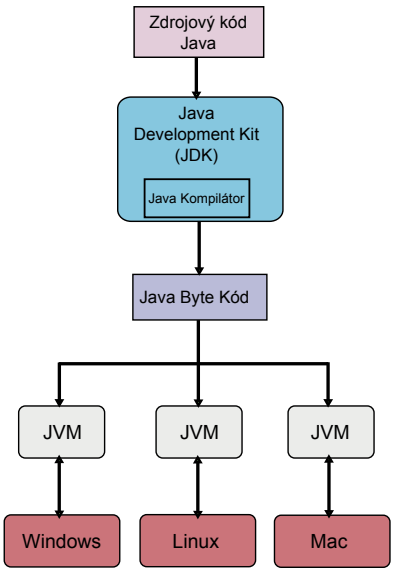
4.4.1 Java platformy

Všechny Java platformy obsahují Java Virtual Machine (JVM) a aplikační programovací rozhraní API. Java Virtual Machine je program pro konkrétní hardwarovou a softwarovou platformu, na které běží aplikace technologie Java. API je soubor softwarových komponent, které je možné použít k vytvoření dalších softwarových komponent či aplikací. Každá platforma Java poskytuje JVM a API. To aplikacím napsaných na platformě Java umožňuje spustit program ve všech kompatibilních systémech. Další výhodou tohoto řešení je stabilita a bezpečnost aplikace. [6]

4.4.2 Spuštění programů v Javě

Java Virtual machine (JVM) je virtuální zařízení, které je spuštěno v operačním systému počítače. JVM poskytuje aplikaci flexibilní nezávislost na platformě. Na obrázku číslo 4.1 je vidět, jak probíhá spouštění kódu v jazyce Java:

1. Nejprve se musí vytvořit zdrojový kód v jazyce Java. Tyto soubory vytváří programátor a mají příponu *.java*.
2. Třídy jazyka Java se zkompilují do byte kódu. Tyto soubory mají příponu *.class*.
3. Java byte kód zpracovává JVM, který vykonává instrukce na konkrétním operačním systému. [10]



Obrázek 4.1: Spouštění Java programů.

4.4.3 Java EE

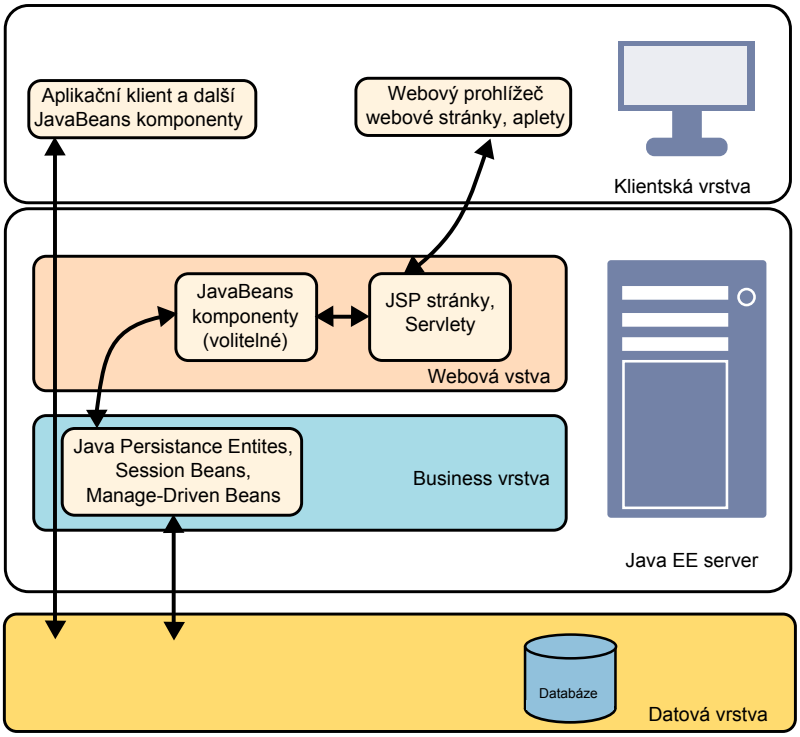
Java Enterprise Edititon (Java EE, dříve označovaná jako Java 2 Enterprise Edition J2EE) je platforma rozšiřující Java SE. Tato platforma slouží především pro vývoj rozsáhlých webových podnikových aplikací. Aplikace obvykle bývají postavené na třívrstvé, nebo čtyřvrstvé architektuře. Java EE poskytuje programátorům sadu užitečných rozhraní, které především šetří čas potřebný pro vývoj aplikací. Také redukují složitost a zlepšují výkon.

Serverové aplikace psané v Java EE běží ve speciálním prostředí nazývaném Aplikační server. Tento server se skládá z kontejnerů a komponent:

- **kontejner** – část aplikačního serveru, která poskytuje prostředí pro běh komponent. Také představuje rozhraní mezi komponentou a jejím okolím,
- **komponenta** – jednotlivá část aplikace, kterou programátor vyvíjí pomocí aplikačních technologií. (Java applety, JavaServer Pages, EJB komponenty atd.).

Ačkoliv je Java EE postavena na třívrstvé architektuře, na stránkách Javy EE má tato architektura vrstvy čtyři[6]. Tuto čtyřvrstvou architekturu je možné vidět na obrázku 4.2.

Další výhodou Javy EE je podpora celé řady frameworků, které programátorovi ulehčují práci. V aplikaci bude použit převážně Spring framework, který podporuje vývoj v Java EE. Tento framework je popsán v následující podkapitole.



Obrázek 4.2: Java EE koncept.

Klientská vrstva

Klientská vrstva představuje část, která je zpracována na klientském zařízení. Tuto vrstvu není třeba implementovat, jelikož klient této vrstvy je prohlížeč.

Webová vrstva

Webová vrstva již běží na Java EE serveru. Ta je tvořena komponentami jako Java Server Pages(JSP), Servlety, případně komponentami Java Beans. V této vrstvě se určuje vzhled a struktura stránky.

Business vrstva

Business vrstva obsahuje veškerou logiku aplikace. Také komunikuje s datovou vrstvou a zajišťuje perzistenci dat. Obecně tato vrstva bývá nejkomplexnější. Proto největší část aplikačního serveru obsluhuje komponenty, které se nacházejí na této vrstvě.

Datová vrstva

Tato vrstva slouží pro práci s relačními databázemi. Rozhraní pro komunikaci může být v Java EE pomocí JDBC. [6]

4.5 Úvod do Spring frameworku

Spring framework je jednoduché řešení pro vytváření enterprise aplikací. Tento framework je modulární, což znamená, že je možné využívat jen ty funkce, které jsou potřebné. Není tedy nutné využívat celý framework, ale jen jeho části. To znamená, že doménová logika kódu obecně nemá závislosti na framework samotný. V integrační vrstvě (například vrstva přistupující k datům) budou existovat některé Spring knihovny v závislosti na technologii přístupu k datům. Nicméně Spring slibuje, že by mělo být jednoduché izolovat závislosti od zbytku kódu.

Spring framework je založen na platformě Java, která poskytuje komplexní podporu infrastruktury pro vývoj aplikací v jazyce Java. Spring řeší infrastrukturu a poskytuje programátorovi možnost zaměřit se přímo na aplikaci. [3]

4.5.1 Vkládání závislosti a Obrácené řízení

Java aplikace velmi obecně řečeno přemosťují omezené aplety na n-vrstvou stranu serveru entrprise aplikací. Obvykle se skládají z objektů, které spolu navzájem spolupracují a umožnují tak řádný chod aplikace. Tyto objekty pak na sebe mají závislosti. Přestože platforma Java poskytuje širokou škálu funkcí pro vývoj aplikací, postrádá prostředky, jak organizovat základní stavební kameny do souvislého celku. Tento úkol nechává pro architekty a vývojáře. Je pravdou, že je možné využít návrhové vzory jako například továrny, abstraktní továrny a různé třídy a instance objektů, které tvoří aplikaci. Vzory jsou formalizované osvědčené postupy, které musí programátor implementovat ve své aplikaci.

Spring framework IoC reaguje na tyto problémy tím, že poskytuje formalizované skládání nesourodých komponent do plně funkční aplikace připravené k použití. Spring framework systematicky třídí návrhové vzory jako first-class objekty, které mohou být integrovány do vlastních aplikací.

Jak již bylo řečeno, Spring framework se skládá z prvků uspořádaných do modulů. Těchto modulů je okolo dvaceti a jsou seskupeny do Core Containeru, Data Access/Integration, Web, AOP, Instrumentation a testování. [3]

4.6 Úvod do Spring Web MVC

Architektura Spring framework Web model-view-controller (MVC) je navržena pro *DispatcherServlet*. Tento servlet předává požadavky handlerům a umožnuje jejich konfigurovatelné mapování, volbu zobrazovacích rozlišení, volbu lokalizačních a tematických rozhraní a také nahrávání souborů. Výchozí handler je založen na anotacích *@Controller* a *@RequestMapping*, které poskytují širokou škálu flexibilních handlovacích metod. S příchodem Springu verze 3.0 dovoluje mechanismus *@Controller* uživateli vytvořit s pomocí *@PathVariable* anotací tzv. RESTful webové stránky a aplikace.

Zobrazovací rozlišení je ve Springu extrémně flexibilní. Controller je obvykle zodpovědný za přípravu modelové mapy (Map) s daty a vybraným pohledem, ale umí

také přímo zapisovat do response streamů a dokončit tak požadavek. Rozlišení při zobrazování jmen je široce konfigurovatelné díky příponě souboru, obsahovému záhlaví, názvům tzv. Beans, konfiguračnímu souboru, nebo také díky nastavitelné implementaci ViewResolveru. Model (písmeno M ve zkratce MVC) je mapovací rozhraní, jenž umožňuje kompletní abstrakci zobrazovací technologie. Je možné přímé integrování s renderovacími technologiemi založenými na JSP šablonách, nebo přímém generování XML, JSON, a spoustu dalších. Modelová mapa je jednoduše transformována do příslušného formátu. [3]

4.6.1 Výhody Spring Web MVC

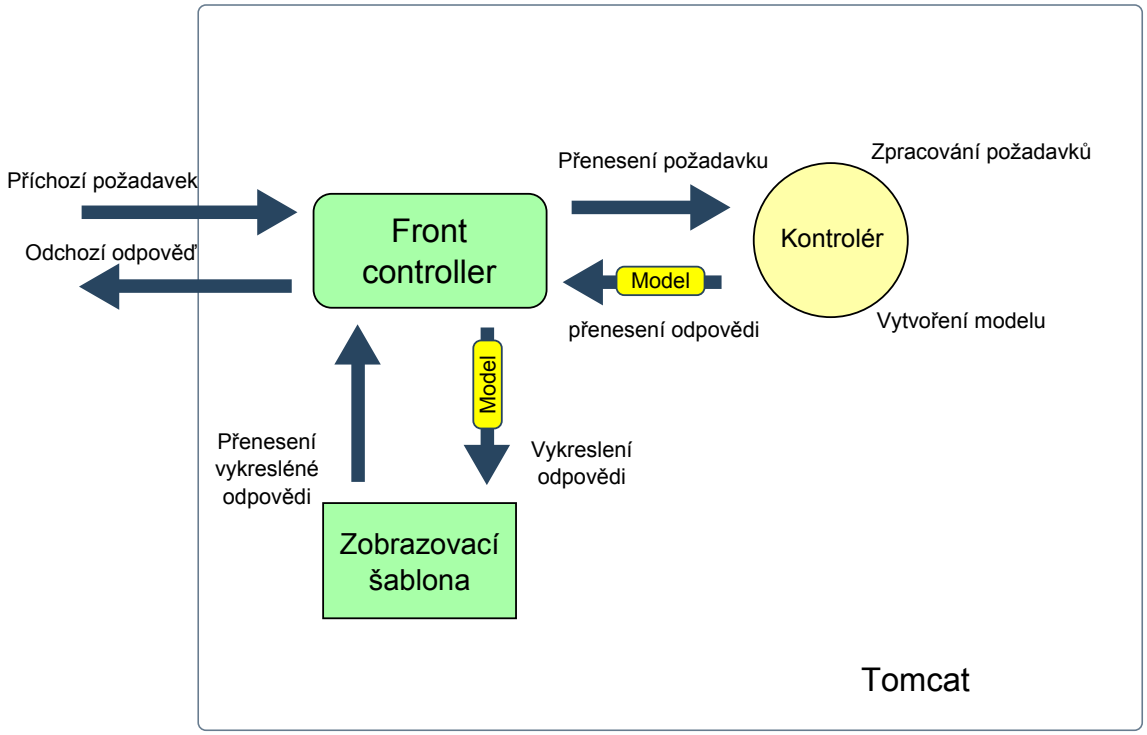
Modul Springu (Spring web module) obsahuje spoustu unikátních příslušenství pro podporu webu:

- čisté oddělení rolí – Každá role – controller, validator, příkazový objekt, formulářový objekt, modelový objekt, DispatcherServlet, handler mapping, view resolver, atd. – může být naplněna speciálním objektem,
- silná a přímá konfigurace tříd frameworku a aplikačních tříd pomocí JavaBeans,
- adaptabilita a flexibilita – Definuje podpis jakékoliv metody kontroléru. S využitím jednoho z anotačních parametrů (např. @RequestParam, @RequestHeader, @PathVariable, atd.) pro daný scénář,
- znovupoužitelnost pracovního kódu – Používá již existující pracovní objekty jako příkazy nebo formulářové objekty, namísto jejich zrcadlení za účelem rozšíření jejich třídní báze,
- nastavitelné vazby a validace - Typové neshody, jakožto chyby aplikační úrovně poskytující problémové hodnoty, lokalizované datum a číslo vazby,
- nastavitelné lokalizace a témata, podpora JSP s knihovnami Spring tagů (nebo i bez knihoven), podpora JSTL,
- jednoduchá, ale silná knihovna JSP tagů známá též jako Spring tag library, poskytující příslušenství jako např. svazování dat a témat. Nastavitelné tagy umožnují maximální flexibilitu při zvyšování hodnoty kódu,
- knihovna JSP tagů, představená ve Spring 2.0, umožnuje mnohem snazší psaní formulářů na JSP stránkách. [3][1]

4.6.2 DispatcherServlet

Spring web MVC framework je jako mnoho ostatních webových MVC frameworků řízen požadavky. Jeho návrh spočívá v centrálním Servletu, který zasílá požadavky ke kontroléru a nabízí další funkcionality, které usnadňují vývoj webových aplikací. DispatcherServlet však znamená víc než jen to. Je kompletně integrovaný s Spring IoC kontejnerem a jako takový umožňuje využívat ostatní funkce, které Spring má.

Front Controller je návrhový vzor, který využívá Spring Web MVC a mnoho dalších webových frameworků. Front controller ve Spring frameworku znázorňuje dipatcherServlet. To znamená, že logické skupiny přijímá jeden logický controller tzv. dispatcherServlet. Má na starosti celý životní cyklus uživatelských požadavků. Cyklus je znázorněn na obrázku 4.3.



Obrázek 4.3: Princip MVC.

Nejprve dispatcherServlet (Front controller) přijme příchozí požadavek. Například url zakončené /Devices. Pomocí třídy HandlerMapping dispatcherServlet zjistí, kterému kontroléru má požadavek předat. V tomto případě by požadavek zpracoval DevicesController. Ten zpracuje požadavek a pošle nazpět objekt ModelAndView, ve kterém je obsažen název zobrazovací šablony a požadovaná data. Název zobrazovací šablony nejprve zpracuje ViewResolver, který navrátí cestu pro view. DispatcherServlet pošle výslednou odpověď (stránku) zpět do webového prohlížeče. [3]

4.7 Konfigurační XML soubory založeny na schématech

V této práci se konfigurace beans provádí přes XML soubory. Z pohledu IoC kontejneru je všechno bean. Velkou výhodou je, že s každou takovou bean je zacházeno přesně stejným způsobem. To ale neplatí z pohledu vývojáře. Všechny objekty definované v konfiguračním XML souboru nemusí být generické. Obvykle každá bean vyžaduje určitý stupeň specifické konfigurace. Konfigurace XML založené na schématech řeší tento problém. Zápis založený na schématech může vypadat takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/
    spring-beans-3.2.xsd">
zde je ~<bean/> definice ...
</beans>
```

Výpis 1: XML konfigurace.

Při zápisu založeném na schématech je možné využívat různé typy schémat. Tato schémata se definují pomocí XML jmenných prostorů (xmlns). [3]. Níže je uveden přehled hlavních schémat, která se využívají v této práci:

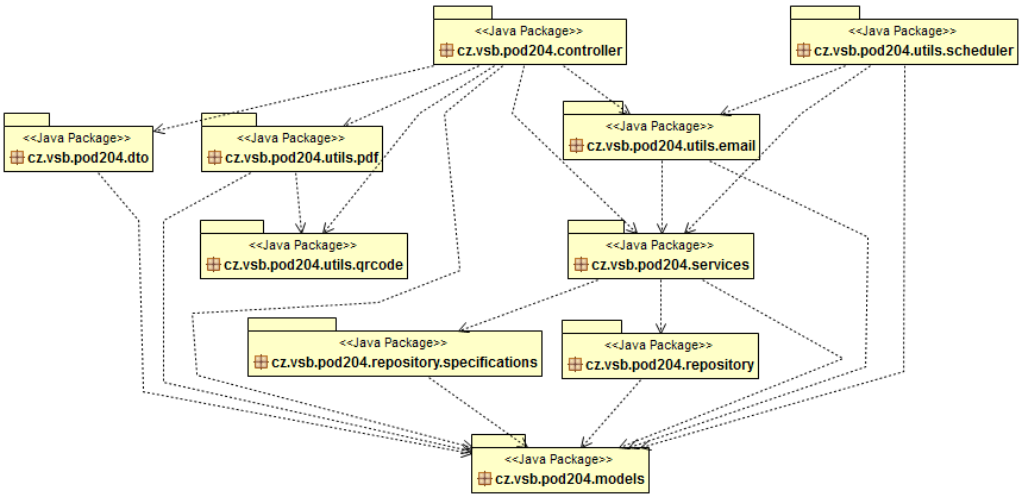
```
xmlns="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:jpa="http://www.springframework.org/schema/data/jpa"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:sec="http://www.springframework.org/schema/security"
```

Výpis 2: Užitá schémata.

4.8 Funkce z pohledu systému

Aplikace obsahuje celkem deset balíčků, které jsou vypsány v tabulce 4.1. Každý balíček vykonává v systému určitou funkci. Nejkomplexnější je balíček *cz.vsb.pod204.controller*, který obsahuje logiku aplikace. Balíčky *cz.vsb.pod204.models*, *cz.vsb.pod204.repository*, *cz.vsb.pod204.repository.specifications*, *cz.vsb.pod204.services* slouží pro přístup k databázi. Balíček *cz.vsb.pod204.utils* slouží k ostatním funkcím jako práci s PDF soubory, či generování QR kódů.

Jak lze vidět na obrázku 4.4, balíčky mezi sebou mají vazby (skrze třídy). Třídy v balíčku *cz.vsb.pod204.controller* zpracovávají všechny události vyvolané URL adresami. Tyto třídy využívají funkce jako zasílání emailů (*cz.vsb.pod204.utils.email*), generování QR kódu (*cz.vsb.pod204.utils.qrcode*) a PDF souborů(*cz.vsb.pod204.utils.pdf*). Také komunikují s databází pomocí služeb z balíčku *cz.vsb.pod204.services*. Třídy z balíčku *cz.vsb.pod204.controller* reagují vždy až na nějakou událost. Systém periodicky kontroluje, zda má uživateli odeslat oznamovací email ke končící lhůtě vypůjčení. O funkce



Obrázek 4.4: Vazby mezi balíčky.

se stará třída v balíčku *cz.vsb.pod204.utils.scheduler*. Dle časového nastavení se služby spouští a ověřují, zda uživatelům nekončí vypůjčovací lhůta. Pokud ano, kontaktují třídy v balíčku *cz.vsb.pod204.utils.email* a odešlou požadované emailové zprávy.

Repositáře v balíčku *cz.vsb.pod204.repository* nekomunikují přímo s kontroléry, ale jsou využívány přes služby v balíčku *cz.vsb.pod204.services*. Díky službám je vrstva k přístupům dat logicky oddělena.

Tabulka 4.1: Balíčky aplikace.

Název balíčku	**Popis balíčku**
cz.vsb.pod204.controller	obsahuje všechny kontroléry
cz.vsb.pod204.dto	uchovává Data Transfer Objekty
cz.vsb.pod204.models	obsahuje všechny Entity
cz.vsb.pod204.repository	zahrnuje všechny repositáře
cz.vsb.pod204.repository.specifications	obsahuje spec. pro *dotazování*
cz.vsb.pod204.services	uchovává služby pro přístup
cz.vsb.pod204.utils.email	zahrnuje třídy sloužící k odesílání emailů
cz.vsb.pod204.utils.pdf	obsahuje třídy sloužící k obsluze PDF
cz.vsb.pod204.utils.qrcode	zahrnuje třídy pro generování QR kódů
cz.vsb.pod204.utils.scheduler	obsahuje třídy pro plánované akce

4.9 Implementace přístupu k databázi

Předtím, než byla zavedena Java Persistance API (JPA) byly tři následující technologie, které bylo možné využít k implementaci persistentní vrstvy:

- persistentní řešení poskytované Enterprise JavaBeans (EJB) 2.X specifikace,
- JDBC API,
- ORM frameworky jako Hibernate.

Široké možnosti implementace perzistentní vrstvy dávaly programátorovi určitou svobodu při výběru nejlepších nástrojů pro práci, ale žádná z těchto možností nebyla bezproblémová.

Problémem EJB 2.X byla příliš náročná konfigurace, která se spoléhala na komplikované XML dokumenty a programovací model vyžaduje velké množství redundantního kódu. EJB také vyžaduje, aby byla aplikace nasazena na Java EE aplikační server.

Programování pomocí JDBC API je poměrně jednoduché a je možné nasadit aplikaci v kterémkoli servlet kontejneru. Nicméně je potřeba napsat velké množství redundantního kódu, který je potřeba při transformaci doménového modelu na dotazy, nebo vytváření doménového modelu z výsledků dotazů.

ORM frameworky třetích stran byly často dobrou volbou, protože programátora osvobozují od psaní zbytečného kódu, který byl použitý k vytvoření dotazů, nebo vytvoření doménových objektů z výsledků dotazů. Tato výhoda má ovšem jeden nedostatek. Objekty a relační data nejsou kompatibilní, a přestože ORM frameworky řeší většinu problémů způsobných objektově-relační neshodou, ty které nelze řešit efektivně jsou nejproblematičtější.

Java Persistence API poskytuje standardní mechanismus pro implementaci perzistentní vrstvy, která používá relační databáze. Hlavní motivací bylo nahradit mechanismus EJB 2.X a poskytnout standardizovaný přístup pro objektově relační mapování. Mnoho vlastností bylo původně představeno ORM frameworky třetích stran, které se později staly implementací Java Persistance API. [2]

4.9.1 Entity

Entita je trvalý doménový objekt. Zpravidla je entita reprezentována tabulkou v relační databázi. Každá instance v entitě odpovídá řádku v této tabulce. Primárním účelem entity je třída entity, i když entity mohou obsahovat pomocné třídy.

Požadavky Entity třídy:

- třída musí být označená @Entity anotací z javax.persistence.Entity,
- musí mít veřejný, nebo chráněný konstruktor bez argumentů. Třída může mít i další konstruktory,
- třída a metody třídy nesmí být deklarovány jako finální,

- pokud session bean bude pracovat s instancemi třídy a zároveň bude typu Remote, potom tato entitní třída musí implementovat rozhraní Serializable,
- entity mohou dědit z entitní i ne-entitní třídy,
- atributy Entit musí být deklarovány jako privátní, chráněné nebo chráněné v balíčku. K atributům lze přistupovat pouze přes metody tj. gettery a settery.[6]

Jednoduchý příklad entity, která reprezentuje tabulku producers viz níže:

```
@Entity
@Table(name="producers")
@NamedQuery(name="Producer.findAll", query="SELECT p FROM Producer p")
public class Producer implements Serializable {
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private int id;
```

Výpis 3: Deklarace entity.

Kód výše před deklarováním třídy obsahuje tři anotace:

- @Entity – označení entity třídy,
- @Table – v anotaci Table se pomocí parametru name definuje název tabulky, kterou entita reprezentuje,
- @NamedQuery – definuje dotaz, jehož názvem je Producer.findAll, který načte všechny objekty Producer, které se nacházejí v databázi.

Objekty uvnitř Entity třídy mohou mít určité anotace. Příkladem těchto anotací jsou:

- @Id – unikátní identifikátor (primární klíč).
- @GeneratedValue – je v tomto případě aplikovaná na primární klíč. Atribut strategy označuje, že by poskytovatel persistence (persistence provider) měl vybrat automaticky vhodnou strategii pro konkrétní databázi.[2][6]

4.9.2 Entity Manager Factory

Továrna Entity Manažera (Entity Manager Factory) vytváří instance Entity manažera. Všechny instance vytvořené stejnou továrnou využívají tutéž konfiguraci a databázi. Pokud aplikace vyžaduje připojení k více než jedné databázi, musí se nakonfigurovat stejné množství továren, jako je počet databází.

Entity manažer řídí Entity v aplikaci. Entity manažer může být používán k provedení tzv. CRUD (CREATE, READ, UPDATE, DELETE) operaci a spouštět komplexní dotazy oproti databázi. Metody Entity Manažera jsou deklarovány rozhraním EntityManager.

Persistentní jednotka (persistence unit) specifikuje všechny Entity třídy, které jsou řízeny Entity Manažerem. Každá persistentní jednotka obsahuje všechny třídy reprezentující data uložená v jedné databázi.

Persistentní kontext obsahuje instanci entity. Uvnitř persistentního kontextu musí být jen jedna instance entity pro všechny identifikátory objektů. Všechny kontextové persistence jsou spojeny s konkrétním Entity manažerem, který spravuje životní cyklus instance entity. [3] [2]

4.9.3 Vytváření databázových dotazů

Persistentní Java API představuje dvě nové metody pro vytvoření databázových dotazů:

- JPQL – Java Presistance Query Language,
- Criteria API.

Dotazy psané těmito technologiemi nepracují přímo s databázovými tabulkami. Místo toho jsou dotazy psány přímo na úrovni Entit. To zajišťuje, že vytvořené dotazy jsou přenosné a nejsou vázány na konkrétní databázové schéma, nebo databázový systém. [3] [2]

4.9.4 JPQL

Java Persistance Query Language je založen na řetězcově dotazovacím jazyce se syntaxi připomínající SQL. Zde je uveden příklad JPQL dotaz z praktické části:

```
"select d from Device d WHERE d.deleted = 1"
```

Tento JPQL dotaz vypíše všechna zařízení, která mají atribut deleted = 1. Je zde také vidět, že se v JPQL dotazuje pomocí odkazu na Entitu Device, nikoliv na tabulku v databázi device.

4.9.5 Spring a práce s JPA

Spring JPA v balíčku org.springframework.orm.jpa nabízí komplexní podporu pro Java Persistance API podobně jako Hibernate či JDO.

Spring nabízí tři možnosti, jak nastavit EntityManagerFactory, které může aplikace použít pro získání objektového (entity) manažera.

Z těchto tří byl vybrán LocalContainerEntityManagerFactoryBean, který poskytuje nejsilnější možnosti nastavení, což umožnuje flexibilní lokální konfiguraci v rámci aplikace. Podporuje připojení na již existující zdroj dat, dále lokální i globální transakce a podobně. Od verze 3.1 již není třeba mít soubor persistence.xml, jelikož Spring nabízí možnost konfigurace pomocí příkazu packageToScan automatickou detekci @Entit. [3] [2]

Deklarativní řízení transakcí

Deklarativní způsob řízení transakcí funguje ve Springu pomocí anotace *@Transactional*. Tento způsob řízení neřeší jednotlivé kroky transakce, to přenechává Spring kontejneru. Výhodou je, že tato volba má nejmenší dopad na kód aplikace.

Pro správný chod transakcí je nutné nastavit v konfiguračním souboru Springu tyto parametry:

```
<tx:annotation−driven transaction−manager=”transactionManager” />.
<beans:bean id=”transactionManager”
      class=”org.springframework.orm.jpa.JpaTransactionManager”>
      <beans:property name=”entityManagerFactory” ref=”myEmf” />
</beans:bean>
```

Výpis 4: Příklad deklarace transakce.

K provádění operací během transakce je možné použít ve Spring JPA frameworku třídu *org.springframework.orm.jpa.JpaTransactionManager*.

Tento správce transakcí je vhodný pro aplikace, které používají jednotnou JPA továrnu entity manažera pro transakční přístup k datům.

Správce transakcí podporuje také přímý přístup ke zdrojům dat v rámci transakce. To umožňuje spojení služeb, které mají přístup skrze JPA společně se službami, které využívají běžný JDBC. V této práci je spojení služeb nezbytné, jelikož komunikace a dotazování probíhají pomocí JPA, ale správa přístupu pomocí knihovny Spring-Acl využívá pouze JDBC. [8] [3]

Po správné konfiguraci je možné použit anotaci *@Transactional* na třídy, či metody. Spring tak automaticky rozpozná, která metoda/třída využívá transakce. Příklad použití anotace *@Transactional* na metodu je možné vidět níže:

```
@Transactional
public Producer create(Producer paramProducers) {
   return producerRepository.save(paramProducers);
}
```

Výpis 5: Příklad použití anotace @Transactional.

4.9.6 Spring Data JPA

Centrální uložiště ve Spring data je repositář. Repozitáře slouží k vykonání atomických operací nad tabulkami. Ve *Spring-Data* Repozitář spravuje doménovou třídu a typ id třídy.

Spring Data JPA je projekt, který má za cíl usnadnění vytváření JPA repositářů a snížení množství kódů potřebné pro komunikaci s databází. Repositáře jsou navrženy tak, aby ulehčily programátorovi práci. Nejprve je nutné repositáře správně nastavit. To se provádí ve Spring konfiguračním xml souboru:

```
<jpa:repositories base-package="cz.vsb.pod204.repository" />
```

Pomoci tagu *jpa:repositories* s atributem *base-package* je nutné nastavit, ve kterém balíčku jsou definovány JPA repositáře. Dále není nutné nějakým způsobem JPA repositáře definovat, Spring se o vše ostatní postará sám.

Modul JPA podporuje tři typy definování dotazů:

- generováním dotazu z názvu metody,
- pomocí @NamedQuery v entity třídě,
- pomocí anotace @Query v repozitáři.[2]

V této práci je využívána anotace @Query, která se definuje v repositářích.

```
public interface ProducerRepository extends JpaRepository<Producer, Integer>{
    @Query("select p from Producer p where p.title = (:name)")
    public Producer findByName(@Param("name") String name);
}
```

Výpis 6: Příklad použití Custom Query.

V kódu uvedeném výše lze vidět rozhraní *ProducerRepository*. Repositář dědí z rozhraní JpaRepository <Entita, typ ID>. V rozhraní je metoda findByName. Tato metoda vyhledá objekt Producer, který má jméno stejné, jako textový řetězec name. Textový řetězec se do JPQL dotazu vkládá pomocí parametru *@Param*. Nad metodou je zmíněná anotace *@Query*, ve které je JPQL dotaz.

4.10 Data Transfer Objekty

Data Transfer Objekty (DTO) přenáší data mezi procesy. V práci jsou použité především pro formulářová data. Například při vytváření zařízení musí uživatel zadat informace jako název, inventární číslo a také zvolit výrobce, který zařízení vyrobil. Celý objekt výrobce ale nemůže být do view přenesen (pouze jeho název či id). V provázanosti na Spring formuláře je nutné, aby každé vstupní pole odkazovalo na objekt (resp. setter a getter). To však není možné, jelikož model zařízení vyžaduje celý objekt výrobce, nikoliv jen id (nebo název výrobce). Vytvořením DTO objektu můžeme nastavit přesně ty informace, které formulář vyžaduje a až následně zkonvertovat DTO objekt do modelu.[16] Také je možné provést validaci některých povinných atributů. V této práci je využita knihovna *org.hibernate.validator*. Příklad takové validace viz níže:

```
public class UserDTO {
    @Email
    private String email;
    @NotEmpty
    @Length(min=4, max=20)
    private String username;
    @NotEmpty
    @Length(min=4, max=20)
    private String password;
```

Výpis 7: Příklad validace v DTO.

Validace se provádí pomocí anotací. V této práci jsou využité tři typy validačních anotací:

- **@NotEmpty** – určuje, že pole nesmí být prázdné,
- **@Length** – definuje minimální a maximální délku řetězce,
- **@Email** – kontroluje, zda má textový řetězec emailovou strukturu.

4.11 Zabezpečení systému

IS využívá pro zabezpečení systému knihovnu *Spring-security* a *Spring-security-acl*. Nejprve se musí nastavit filtr v souboru web.xml. Třída, která spravuje tento filtr je *org.springframework.web.filter.DelegatingFilterProxy*. Tímto se v aplikaci vytvoří proxy, která zpracovává všechny požadavky definované v elementu: *url-pattern*. Následně se nastaví Spring-Securitity konfigurační XML soubor:

```
<http use-expressions="true">
<form-login
login-page="/login"
authentication-failure-url="/login?login_error=1"
   default-target-url="/Overview" always-use-default-target="true"
/>
```

Výpis 8: Nastavení Spring-Security konfiguračního souboru.

Pro zabezpečení jednotlivých URL je nutné nastavit v http tagu atribut *use-expressions="true"*. Následně se nastaví formulářová autentizace pomocí tagu *form-login*. V tomto tagu se atributem *login-page* nastaví stránka, která nepřihlášeného uživatele přesměruje na přihlašovací stránku. Atribut *authentication-failure-ur*l přesměruje uživatele v případě nesprávného přihlášení na chybovou stránku. Také je nutné nastavit stránku, na kterou bude uživatel přesměrován po úspěšné autentizaci. K tomu slouží tag: *default-target-url*. Uvnitř http tagu se nastaví, které role jsou oprávněny přistoupit k dané URL adrese pomocí:

```
<intercept-url pattern="/SuperAdmin/Settings/*"
access="hasRole('ROLE_SUPERADMIN')" />
```

Kdy *pattern* je vzor URL adresy, o kterou se jedná a atribut *access* určuje, která role k URL může přistoupit.

Zabezpečení pomocí URL je dobrým nástrojem pro zabezpečení statických URL adres. V případě, že se používá *PathVariable*, tedy dynamické URL cesty, tak toto zabezpečení není vhodné. Proto je zabezpečení URL v práci použité pro statické stránky, jako například vytvoření zařízení, či administrace. Pro případy, kdy nestačí zabezpečení skrze URL se ve Spring-Security používají anotace rovnou nad metody, ke kterým by měl být ověřen přístup. Před nastavením anotací je třeba v konfiguračním souboru Springu povolit v elementu *global-method-security* pre a post anotace:

```
<sec:global-method-security pre-post-annotations="enabled" />
```

Jakmile jsou anotace povoleny, je možné je použít přímo nad metodami. Příklad anotace je možné vidět níže:

```
@PreAuthorize("hasPermission(#device, admin)
            or hasAnyRole('ROLE_ADMIN', 'ROLE_SUPERADMIN')")
    public void update(Device device) {
        deviceRepository.save(device);

    }
```

Výpis 9: Deklarace přístupu v Java kódu.

Anotace *@PreAuthorize* se před voláním metody dotazuje, zda má uživatel právo ke spuštění dané metody. V tomto případě probíhá ověření dvěma metodami:

- *hasPermission* – kontrola přístupových listů, zda má uživatel pro objekt device administrátorské právo.
- *hasAnyRole* – tato metoda kontroluje, zda má uživatel v systému určitou roli (v tomto případě ADMIN, nebo SUPERADMIN).

Mezi těmito dvěma metodami je logická disjunkce. Pokud tedy uživatel splňuje alespoň jednu z podmínek, je mu přístup udělen. [1] [4] [11]

4.11.1 Práce s přístupovými listy

Nejprve je třeba rozebrat tabulky, které využívá *Spring-Security-Acl*. Ty jsou stručně rozepsány v následujících řádcích.

ACL_SID

Umožňuje jednoznačnou identifikaci všech uživatelských identit nebo autorit v systému. Tabulka obsahuje pouze tři sloupce:

- unikátní identifikátor,
- textový řetězec reprezentující SID,
- ukazatel, zda se jedná o uživatelskou identitu, nebo udělenou autoritu.

Jeden řádek obsahuje unikátní uživatelskou identitu nebo udělenou autoritu. V souvislosti s obdržením oprávnění je SID obecně nazýván jako „příjemce“.

ACL_CLASS

Umožnuje jednoznačně identifikovat všechny třídy doménových objektů v systému. Tato tabulka obsahuje dva sloupce:

- unikátní identifikátor,
- název Java třídy.

Jeden řádek obsahuje unikátní Java třídu, kterou si přejeme uložit v ACL přístupech.

ACL_OBJECT_IDENTITY

Ukládá informace o všech unikátních instancích doménových objektů v systému. Tato tabulka obsahuje sloupce:

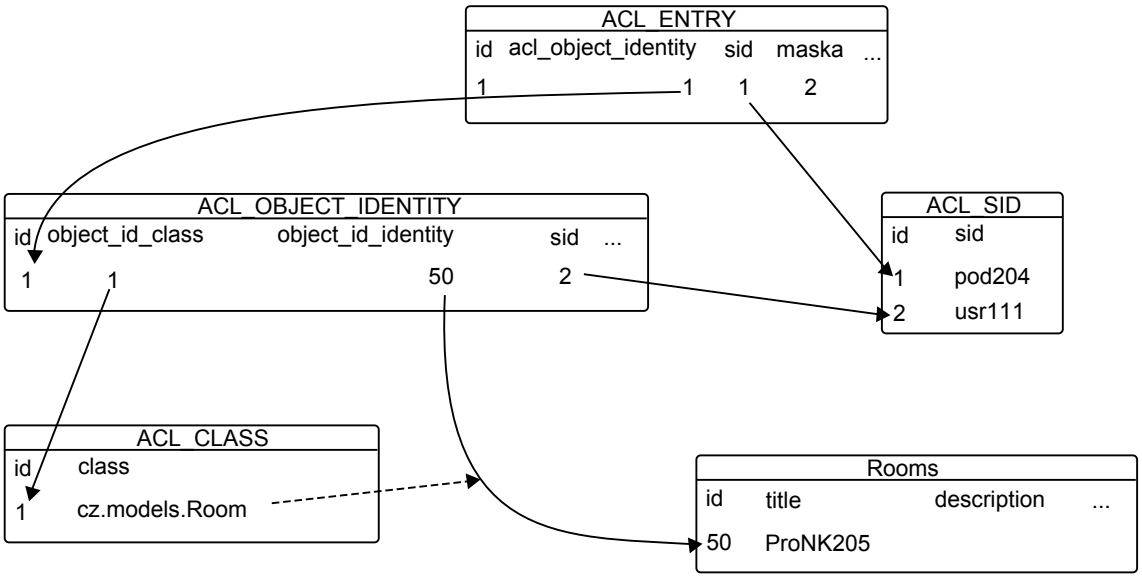
- unikátní identifikátor,
- cizí klíč na tabulku ACL_CLASS - unikátní identifikátor, díky kterému je možné rozpoznat instanci třídy,
- rodič,
- cizí klíč na tabluku ACL_SID – která reprezentuje vlastníka instance doménového objektu,
- inherit – označuje, zda ACL položka dědí z jakékoliv mateřské ACL.

ACL_ENTRY

ACL_ENTRY ukládá individuální přístupy pro všechny příjemce a obsahuje tyto sloupce:

- unikátní identifikátor,
- cizí klíč na ACL_OBJECT_IDENTITY,
- cizí klíč na ACL_SID,
- informaci, zda tento přístupový list povoluje, či zamítá přístup,
- bitovou masku Integer reprezentující aktuální přístup (ať udělen, či zamítnut),
- pořadí, ve kterém jsou zpracována pravidla,
- příznaky, zda auditovat úspěšné/neúspěšné povolení. [1] [4]

Na obrázku 4.5 jsou vidět vazby mezi tabulkami. Tento vzorový příklad se týká vytvoření přístupových listů k místnosti. Tabulka ACL_ENTRY odkazuje na ACL_OBJECT_IDENTITY a na tabulku ACL_SID, kde se určuje, kterému uživateli budou přidány přístupová práva. ACL_OBJECT_IDENTITY nese referenci na třídu a id třídy v tabulce, které objekt náleží. V tomto případě se jedná o třídu *cz.models.Room* a řádek v tabulce s id = 50. ACL_OBJECT_IDENTITY také obsahuje informaci o tom, kdo záznam vytvořil, takže je možné tuto informaci dohledat. V Java kódu je metoda



Obrázek 4.5: Vazby mezi ACL.

definována následovně:

```
private void createAcl(Room room) {
    int roomId = room.getId();
    ObjectIdentity roomOid = new ObjectIdentityImpl(Room.class, roomId);
    MutableAcl roomAcl = (MutableAcl) aclService.createAcl(roomOid);
    Sid author = new PrincipalSid(room.getUser().getUsername());
    roomAcl.insertAce(roomAcl.getEntries().size(), BasePermission.WRITE, author, true);
    aclService.updateAcl(roomAcl);
    }
```

Výpis 10: Vytvoření Access Listu.

Ve výše uvedeném příkladu je načtené ACL spojeno s doménovým objektem *Room* spolu s jeho *id*. Přidáním *ACE* se nastaví uživateli author právo zápisu do místnosti. Posledním argumentem metody *insertAce* určíme, zda je uživateli povolen, či zamítnut přístup. Metodou *updateAcl* se informace o přístupovém listu uloží do databáze.

4.12 Implementace funkcí systému

V kapitole 4.6.2 je nastíněno, jak funguje životní cyklus ve Spring Web MVC. Controller obsahuje všechny důležité funkce. Aplikace celkem obsahuje šest kontrolérů, které je možné vidět v tabulce 4.2.

Tabulka 4.2: Přehled kontrolérů.

Název třídy	Stručný popis třídy
ControllerDevices	Všechny funkce týkající se zařízení
ControllerImages	Funkce zobrazování obrázků
ControllerProducers	Správa výrobců
ControllerRest	Mobilní API
ControllerRooms	Funkce dostupné v místnosti
ControllerWeb	Nastavení serveru, uživatelů a PDF šablon

4.12.1 Implementace funkcí zařízení

V této sekci jsou funkce, které vykonává kontrolér *ControllerDevices*. Tato třída obsahuje celkem dvaadvacet metod, které jsou převážně zaměřeny na zařízení. Metody označené anotací *@RequestMapping* se vyvolají v případě, že uživatel přejde na URL adresu definovanou v této anotaci. HTML formuláře podporují pouze dva typy HTTP požadavku a to GET a POST. HTTP požadavky jsou rozeznány pomocí metod *RequestMethod.GET* a *RequestMethod.POST*. Metoda GET slouží v systému k zobrazení dat a pomocí metody POST se manipuluje s daty. Většina metod třídy *ControllerDevices* je vztažena k jednomu zařízení, které je rozpoznáno podle *id*. Toto *id* je předáno pomocí URL adresy. Systém URL adresu rozpozná pomocí anotace *@PathVariable*.

```
@RequestMapping(value = { "ShowDevice/{id}" }, method = {RequestMethod.GET })
    public String  showDevice(@PathVariable(value = "id") Integer deviceId, Model m) {
        Device device = deviceService.findByDeviceId(deviceId, 0);
        ...
```

Výpis 11: Příklad metody v ControllerDevices.

V případě úspěšného parsování *deviceId* se přejde k vykonání metody. Pokud parsování úspěšné nebude, systém zahlásí chybu se statusem 400 a chybovou hláškou: „*The request sent by the client was syntactically incorrect ()*“. Funkce, které obsahuje *ContrllerDevices* jsou shrnuty do těchto bodů:

- vytváření, editace, zobrazení a smazání zařízení,
- vypůjčení zařízení do jiných místnosti a zápis stavů,
- nahrávání, zobrazení a smazání příloh a obrázků zařízení,
- export do PDF šablony.

Vytváření a editace se zadávají pomocí formulářových dat. Zobrazení velkého množství dat je pro přehlednost uspořádané do tabulkového formátu. U vypůjčených zařízení je možné odeslat notifikační email k vyzvání vrácení zařízení. Nastavení emailového klienta je dostupné v záložce přístupné hlavním administrátorům „Správa systému“. Emaily se posílají pomocí knihovny *javax.mail*. Tato implementace se nachází v balíčku *cz.vsb.pod204.utils.email*.

4.12.2 Generování QR kódů

Pro generování QR kódů systém využívá knihovnu *Zxing* . Tato open-source knihovna umožňuje práci s 1D a 2D čárovými kódy. Knihovna je implementovaná v jazyce Java. QR kódy jsou využívány těmito funkcemi:

- v detailu zařízení (ControllerImages),
- v hromadném exportu zařízení do PDF souboru (ControllerRooms),
- v detailním exportu zařízení do PDF souboru (ControllerDevices).

Třída, která pomocí knihovny Zxing slouží pro generování QR kódů se nachází v balíčku *cz.vsb.pod204.utils.qrcode*. Postup generování je následující:

1. Vytvoří se požadovaný textový řetězec s kódováním nastaveným na UTF-8,
2. Následně se do bitové matice uloží QR kód. V tomto kroku se také nastaví velikost QR kódu, tichá oblast a kódování,
3. Vytvořený QR kód se pomocí třídy *MatrixToImageWriter* a metody toBufferedImage převede do obrázku.
4. V controleru je tento obrázek zpracován podle potřeby.

4.12.3 Export do PDF souborů

Další komplexnější funkcí je export do PDF souboru, která využívá knihovnu *itext*. PDF soubor obsahuje následující položky:

- QR kód v levém horním rohu,
- náhled zařízení (pokud je nahrán),
- název zařízení,
- inventární číslo, podčíslo a sériové číslo,
- webový odkaz na zařízení,
- výrobce,
- domácí místnosti zařízení a jejího správce,

- aktuální místnosti zařízení,
- ceny,
- datum, kdy bylo PDF vygenerováno.

Implementace PDF souborů se nachází v balíčku *cz.vsb.pod204.utils.pdf*. Příklad výstupu PDF souboru je možné vidět na obrázku číslo 4.6.

4.13 Implementace funkcí místnosti

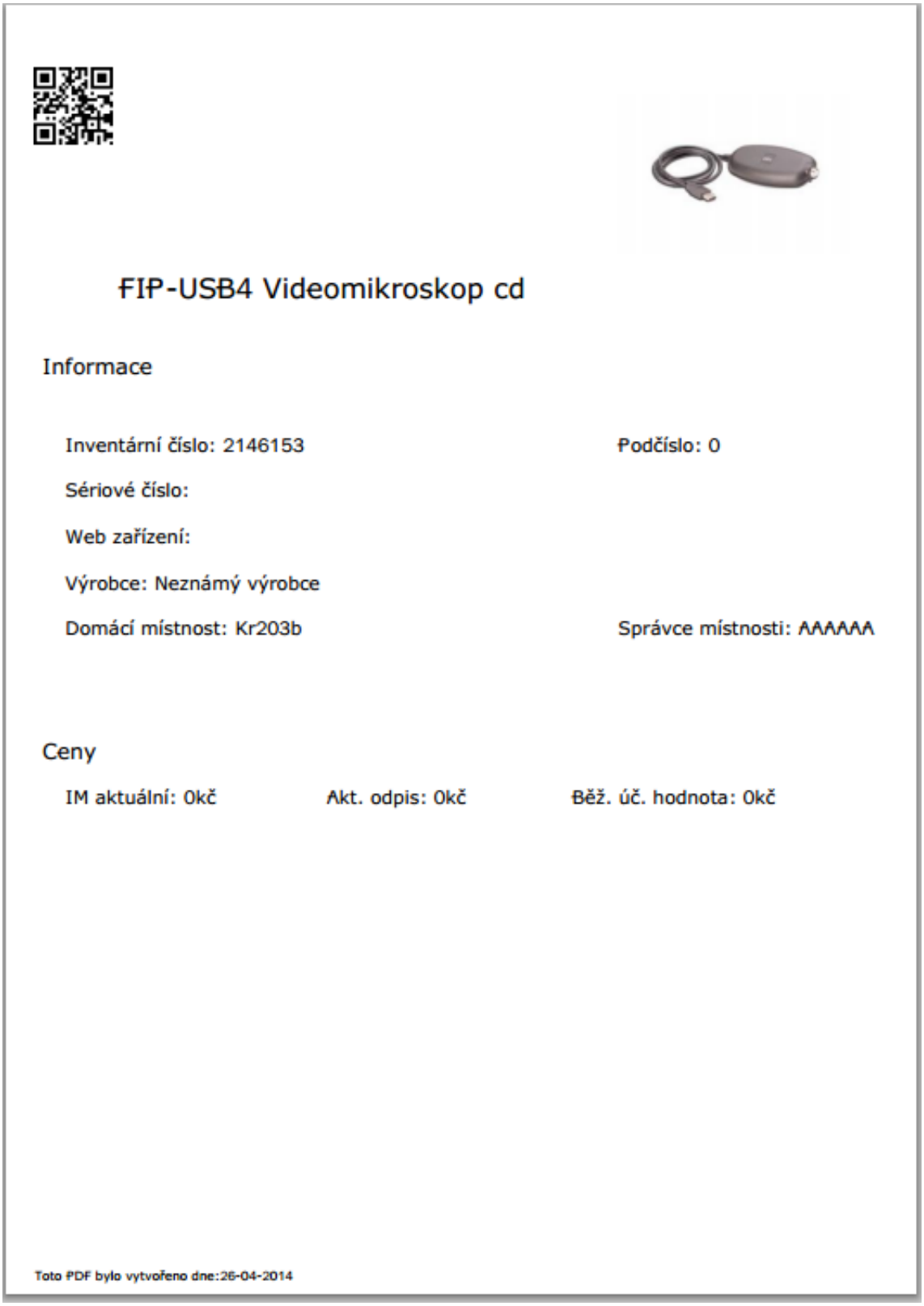
Funkce místnosti obsluhuje *RoomsController*, který se nachází v balíčku *cz.vsb.pod204.controller*. Tento kontrolér obsahuje devatenáct metod. Metody nezpracovávají pouze funkce místnosti jako vytváření, editaci a mazání. Důležitou funkcí v místnosti jsou hromadné importy, exporty a také přesuny zařízení. *RoomsController* zpracovává obecně následující požadavky:

- vytváření, editace, zobrazování a smazání místnosti,
- hromadný import z xls/xlsx souborů,
- hromadné zobrazení/vypůjčení/smazání zařízení v místnosti,
- hromadný export do xls/xlsx souborů,
- hromadný export do csv souborů,
- hromadný export PDF šablon.

4.13.1 Hromadné importy zařízení do místnosti

Hromadný import slouží k rychlému nahraní zařízení do místnosti pomocí souboru xls/xlsx. Systém nejprve zkontroluje, zda má uživatel přístup k hromadnému importu zařízení. Poté si uživatel vybere, jaký typ importu chce provést. K dispozici jsou celkem dvě varianty. První varianta je import z xls/xlsx SAP, který vygeneruje SAP v MHTML souboru. Tento soubor je nutné uložit jako xls/xlsx. Druhou variantou je klasický import z xls/xlsx, který je kompatibilní s exportem a tím rozšiřuje možnosti systému jako například zálohování zařízení. Implementace probíhá u obou importů velmi podobně, rozdíl je pouze u počtů sloupců. V souboru ze SAPu jsou uloženy pouze tyto informace:

- investiční majetek (Inventární číslo),
- podčíslo,
- označení IM (Název zařízení),
- IM aktuální
- aktuální odpis,



FIP-USB4 Videomikroskop cd

Informace

Inventární číslo: 2146153 Podčíslo: 0

Sériové číslo:

Web zařízení:

Výrobce: Neznámý výrobce

Domácí místnost: Kr203b Správce místnosti: AAAAAA

Ceny

IM aktuální: 0kč Akt. odpis: 0kč Běž. úč. hodnota: 0kč

Toto PDF bylo vytvořeno dne:26-04-2014

Obrázek 4.6: Příklad detailu zařízení exportovaného do PDF šablony.

- běžná účetní hodnota.

V SAP souboru chybí informace ohledně výrobce. Proto je každému zařízení, které je importováno z tohoto typu souboru přiřazen Neznámý výrobce. Výrobce zařízení je možné změnit v detailu zařízení. V klasickém importu jsou tato data ještě rozšířena o sloupce:

- Sériové číslo
- Výrobce
- Popis

K práci s xls/xlsx soubory slouží knihovna *org.apache.poi*. Knihovna poi je velmi dobře popsána a názorně vysvětlená na příkladech. [12] Proto zde nebude popsán Java kód, ale logika metody.

1. Vytvoří se instance třídy *Workbooku*, kde je načten nahraný soubor.
2. Z *Workbook* se vybere list(*Sheet*) s indexem 0.
3. Systém postupně prochází dokument řádek po řádku (ignoruje pouze první řádek s hlavičkou).
4. V každém řádku se postupně načítají sloupce, které mají pevně danou strukturu. Každý sloupec obsahuje určitá data, která se uloží do objektu *Device*.
5. Následně probíhá validace objetku *Device*.
   - Zda inventární číslo má délku větší než 4 a zda je číslo.
   - Zda podčíslo je číslo.
   - Zda se v dokumentu nenachází stejná kombinace inventárního čísla a podčísla.
   - Zda se v databázi nenachází inventární číslo a podčíslo.
6. Po validaci se zařízení rozdělí na úspěšně validovaná a neúspěšně validovaná.
7. Úspěšně validovaná zařízení jsou uložena do databáze.
8. V posledním kroku je uživateli zobrazen tabulkový výpis jednotlivých řádků, který je možné vidět na obrázku číslo 4.7. V případě neúspěšné validace je uživateli zobrazen důvod, proč zařízení nebylo nahráno na server.

Obrázek 4.7: Výpis hromadného importu.

4.14 Implementace API pro mobilního klienta

Komunikace s mobilním zařízením probíhá pomocí metody obsažených v HTTP protokolu. Jedinou funkci, kterou mobilní klient využívá je dotazování na detailní informace ohledně zařízení. Dotaz je složen z inventárního čísla a podčísla. Ty se zasílají jako parametr v URL adrese. Metoda, která obsluhuje tuto komunikaci je nazvána *restFindDevice* a její část je popsána níže.

```
    @RequestMapping(value = "/device", method = RequestMethod.GET)
    @ResponseBody
public String restFindDevice(@RequestParam(value="InventoryNumber", required=true) String
    deviceInvNumAndInNum) {
...
```

Výpis 12: Příklad metody v ControllerRest.

Anotace *@RequestMapping* již byla popsána dříve. Druhou anotací je *@ResponseBody*. Díky této anotaci Spring pozná, že je návratová hodnota textový řetězec. V argumentu se nachází další anotace *@RequestParam*. V anotaci je nastaveno, jaký má mít parametr název a zda je povinný či volitelný. Tato metoda přijímá požadavek ve tvaru:

```
/rest/device?InventoryNumber=277651892_0
```

Parametr *InventoryNumber* se skládá z dvou částí. První části je inventární číslo a druhou části je podčíslo. Podčíslo je od inventárního odděleno „_“ znakem. Odpověď je odeslána je v datovém formátu JavaScript Object Notation (JSON). Ke konverzi dat slouží knihovna google-gson. Konverze probíhá pomocí příkazu:

```
gson.toJson(returnedDevice, DeviceRESTDTO.class);
```

Prvním argumentem této metody je objekt s daty. Druhým argumentem je třída, která reprezentuje objekt s daty. Návratova hodnota metody je textový řetězec. Při dotazu mohou nastat tyto situace:

- Zařízení se úspěšně vyhledá v databázi a textový řetězec je navrácen v datovém formátu JSON.
- URL adresa dotazu je ve správném tvaru v databázi se nenachází požadovaná kombinace inventárního čísla a podčísla. V takovém případě systém vypíše chybovou hlášku: *Exception:DeviceNotFound!*
- URL adresa je ve špatném formátu a systém navrátí: *Exception:Inventory number and subnumber is not valid!*

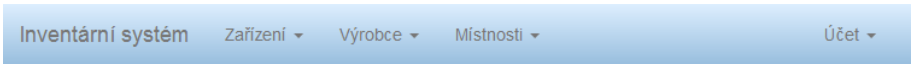
Autentizace neprobíhá pomocí formuláře pro autentizaci, ale pomocí zabezpečení HTTP basic. Toto zabezpečení je poměrně snadno prolomitelné a proto je komunikaci vhodné zašifrovat. Šifrovanou komunikaci nerealizuje aplikace, ani aplikační server, ale reverzní proxy server Nginx. Konkrétní nastavení, které slouží pro šifrovanou komunikaci je uvedeno v příloze na DVD.

4.15 Implementace grafického uživatelského rozhraní

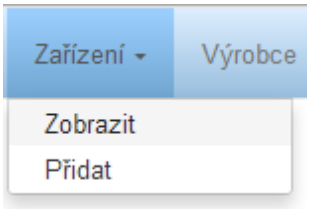
Grafické uživatelské rozhraní slouží pro komunikaci s uživatelem. V kapitole byla 3.4 provedena analýza. Díky této analýze bylo jednoduché vytvořit přívětivé uživatelské rozhraní. V této kapitole jsou příklady jednotlivých prvků, které zobrazuje systému uživateli.

4.15.1 Menu lišta

Menu lišta je zobrazena uživateli až po úspěšném přihlášení do systému. Tuto lištu lze je vidět na obrázku 4.8. Detail podmenu lze vidět na obrázku 4.9.



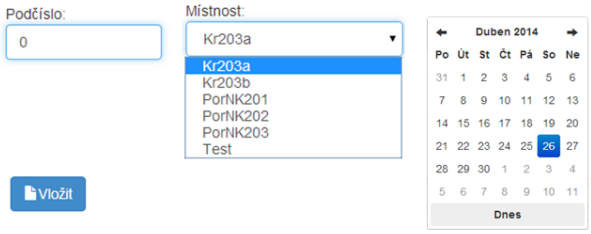
Obrázek 4.8: Menu lišta.



Obrázek 4.9: Podmenu.

4.15.2 Formuláře

Formulářové prvky souží k zadávání informací do systému a jsou zobrazeny na obrázku 4.10.



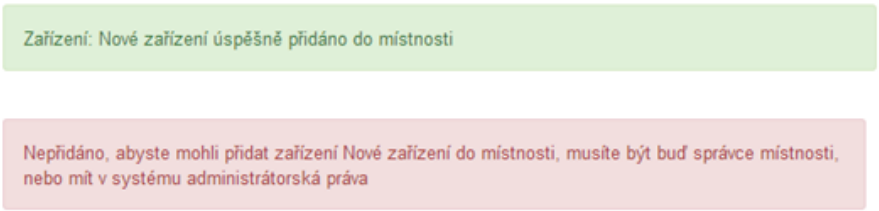
Obrázek 4.10: Formulářové prvky.

Příklad formuláře je zobrazen na obrázku číslo 4.11

Obrázek 4.11: Příklad formuláře.

4.15.3 Zpětná vazba

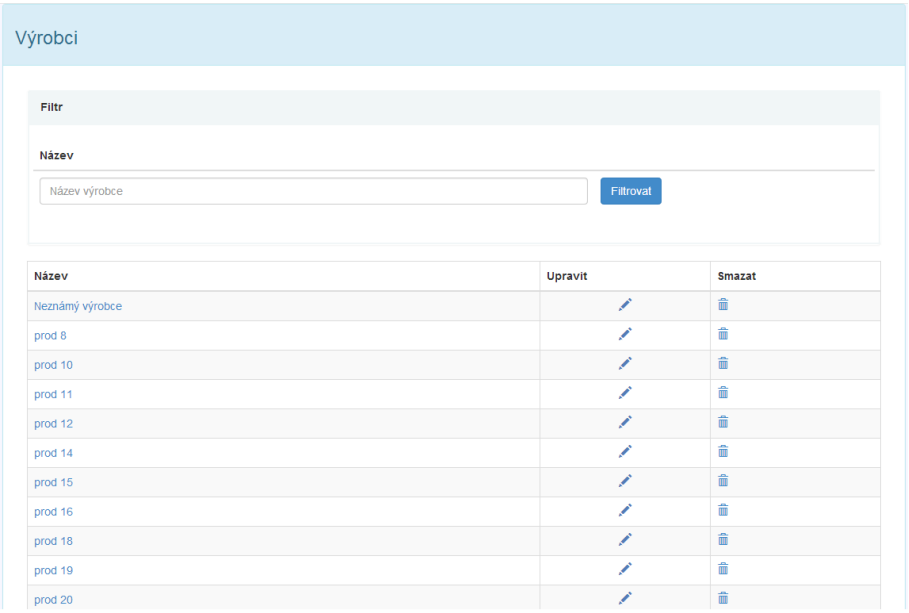
Slouží uživateli jako odpověď na jeho akci. Kladná zpětná vazba je označena zelenou barvou. Červená barva slouží k indikaci záporné odpovědi systému. Ke každé hlášce je zobrazen text. Příklad úspěšné a neúspěšné zpětné vazby je znázorněn na obrázku 4.13.



Obrázek 4.12: Zpětná vazba.

4.15.4 Hromadné výpisy

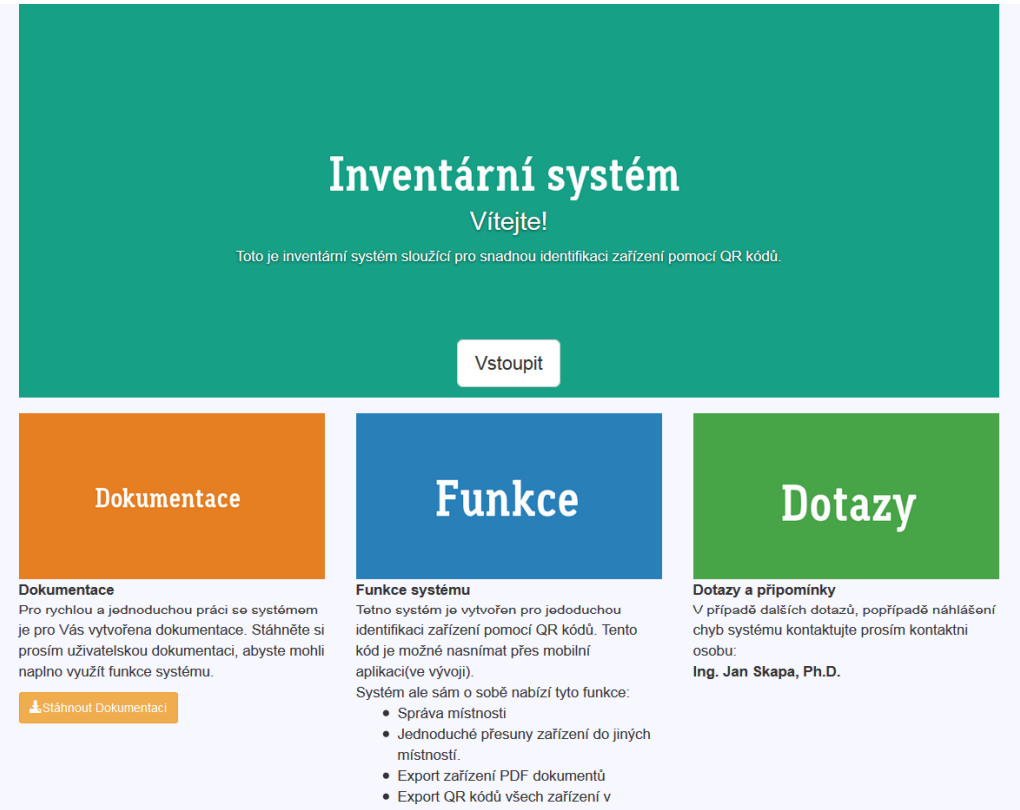
Pro přehlednost jsou všechny liché řádky označeny jinou barvou, než řádky sudé. Výpisy obsahují funkční prvky, které uživateli nabízí další funkce. Pro zobrazení detailu stačí kliknout na název položky, pro úpravu slouží ikona tužky a odstranění záznamů je znázorněno ikonou koše. Příklad zobrazení reportu všech výrobců je znázorněn na obrázku 4.13.



Obrázek 4.13: Zobrazení hromadného výpisu výrobců.

4.15.5 Titulní strana

Titulní strana slouží k rychlému seznámení s aplikací. Jsou v ní popsány funkce systému, dokumentace a kontaktní osoba v případě dotazů. 4.14



Obrázek 4.14: Úvodní strana webové aplikace.

5 Dokumentace a testování

5.1 Dokumentace

Systém je navržen tak, aby byl jednoduchý na ovládání. Přesto je vytvořená uživatelská dokumentace, která slouží pro snadnou orientaci ve webové aplikaci. Tento manuál také napomáhá uživateli k rychlému pochopení systému a jeho funkcí. Vysvětleny jsou také detailní postupy a možnosti systému. Hlavními kapitolami uživatelské dokumentace jsou:

- úvod a základní popis systému,
- zařízení,
- místnosti,
- výrobci,
- uživatelský profil,
- administrátorské funkce,
- funkce hlavního administrátora.

Aby byla zajištěna dostupnost dokumentace každému uživateli, je možné si tuto dokumentaci stáhnout na úvodní straně, která je znázorněna na obrázku 4.14

5.2 Testování

Testování IS je nezbytné pro odhalení chyb a nedostatků systému. Testování bylo provedeno nejprve vývojářem na *localhostu*. Poté byla aplikace nasazena na produkční prostředí, kde byly opraveny spíše minoritní chyby. Aplikace je nasazena na serveru, jehož parametry jsou znázorněny v tabulce 5.1.

Tabulka 5.1: Parametry serveru.

Procesor	Intel Xeon X5560 2.8 Ghz
Paměť	4GB RAM
Velikost disku	40GB
Operační systém	Ubuntu 13.10 x86_64

Na serveru je nastaven opensource aplikační server Apache Tomcat, který je založen na jazyce Java. Informace o serveru jsou v tabulce 5.2.

Tabulka 5.2: Parametry Apache Tomcat.

Verze Tomcatu	7.0.42
Verze JVM	1.7.0_45-b18
JVM vendor	Oracle

Na produkčním prostředí byly také provedeny zátěžové testy, kde bylo do systému celkem nahráno 50 000 zařízení. Všechna zařízení nebyla nahrána přímo do databáze, ale pomocí funkce hromadného importu z xls formátu. Tato zařízení byla importována po dávkách do jednotlivých místností. Počet importovaných zařízení a doba zpracování lze vidět v tabulce 5.3.

Tabulka 5.3: Testování hromadného importu do místnosti.

Počet řádků v xls/xlsx souboru	Doba zpracování [ms]
300	1 450
600	4 120
3000	23 320
6000	75 632

Jak lze vyčíst z tabulky, doba se rapidně zvyšuje s počtem řádku v souboru. To proto, že je nutné provést validaci duplicitních inventárních čísel a podčísel. Tato validace by mohla být lépe optimalizovaná pro větší počet zařízení, ale jelikož je v celé katedře okolo 5000 záznamů, takto velký počet zařízení nebude do jedné místnosti nahrán.

V jednotlivých místnostech byl také proveden hromadný export do xls a csv formátů. Dobu exportu je možné vidět v tabulce 5.4 a tabulce 5.5. v těchto tabulkách jde vidět téměř dvakrát větší rychlost zpracování u csv formátu oproti xls formátu. To může být způsobené tím, že se v metodě export do xls používá metoda sheet.autoSizeColumn();, která má vliv na výkon zpracování. [13] I při dvojnásobné režii je z pohledu uživatele tato rychlost dostačující, jelikož celkový počet evidovaných zařízení ve všech místnostech je okolo 5000.

Tabulka 5.4: Testování hromadného exportu do xls souboru.

Počet zařízení v místnosti	Doba zpracování [ms]
300	820
600	1 750
3000	9 670
6000	17 387

Tabulka 5.5: Testování hromadného exportu do csv souboru.

Počet zařízení v místnosti	Doba zpracování [ms]
300	350
600	850
3000	2 841
6000	6 213

Překvapivým výsledkem je export QR kódů do PDF souborů. Doba exportu je znázorněna v tabulce číslo 5.6. Export do PDF souboru sice nevyžaduje tolik dat, jako je tomu v hromadném exportu, zato ale každý QR kód je převeden do obrázku, které se vloží do PDF.

Tabulka 5.6: Testování hromadného exportu do PDF souboru.

Počet zařízení v místnosti	Doba zpracování [ms]
300	293
600	624
3000	3 121
6000	5 450

6 Závěr

Cílem diplomové práce bylo vytvořit inventární systém, který je možné využít nejen na Katedře telekomunikační techniky. S vyvíjejícími se požadavky se systém stal nejen inventárním, ale také rezervačním. Systém je platformě nezávislý jak z pohledu uživatele, tak z pohledu serveru. To proto, že uživatel přistupuje k systému pomocí webového prohlížeče a serverová část je psaná v programovacím jazyce Java.

Teoretická část čtenáře seznamuje s QR kódy a jejich možnostmi. Tato analýza byla provedena na základě zkušeností s dřívějším systémem a uživatelských zkušeností. Velká část práce je věnována popisu požadavků a jejich teoretickému řešení, které jsou nezbytné pro návrh kvalitního informačního systému.

Systém také není vázán pouze na jeden typ databáze, což umožňuje ještě větší flexibilitu v nasazení do různých prostředí. Datová analýza je navržena obecně a nevztahuje se na konkrétní druh databáze. Pro implementaci byla zvolena jedna z nejrozšířenějších databází MySQL, ke které jsou vytvořené vzorové skripty.

Dalším z požadavků bylo vytvoření vhodného API pro přístup mobilního klienta, které je navrženo pro poskytnutí detailních informací ohledně zařízení. Toto API především slouží k přenesení náročnosti z klientského zařízení na serverovou komponentu. Ve vytvořeném rozhraní použitím jednotného komunikačního protokolu také odpadly problémy způsobené vazbou na konkrétní technologie.

Z důvodu rychlé orientace v aplikaci byla vytvořená uživatelská dokumentace, která slouží pro správné pochopení systému. Také jsou v ní podrobné návody, které uživateli vysvětlují, jak postupovat při jednotlivých úkonech. Přínosem této práce je především informační systém, který poskytuje uživatelům rychlý přehled ohledně zařízení. Správci místností pomocí aplikace získávají podrobný přehled o svých místnostech. Ve spojení s klientskou aplikací systém slouží pro rychlou identifikaci zařízení.

Vývoj aplikace se bude i nadále rozšiřovat o další funkcionality na základě požadavků.

Literatura

[1] WILLIE WHEELER, John Wheeler. Spring in practice. Greenwich, Conn: Manning, 2009. ISBN 19-351-8205-6.

[2] KAINULAINEN, Petri. Spring data. Birmingham: Packt Publishing, 2012, iv, 143 p. ISBN 978-1-84951-904-5.

[3] Spring Framework Reference Documentation. Spring development team WEBB. Spring Framework [online]. [cit. 2014-05-04]. Dostupné z: http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/htmlsingle/

[4] Spring security 3.x cookbook. S.l.: Packt Publishing Limited, 2013. ISBN 978-178-2167-525.

[5] ŠARMANOVÁ, Jana. Databázové a informační systémy. Ostrava: Vysoká škola báňská - Technická univerzita, 2007, 1 CD-R. ISBN 978-80-248-1499-5.

[6] The Java EE 6 Tutorial. ORACLE. Oracle [online]. 2013 [cit. 2014-05-04]. Dostupné z: http://docs.oracle.com/javaee/6/tutorial/doc/docinfo.html

[7] Spring Data JPA - Reference Documentation. GIERKE, Oliver a Thomas DARIMONT. Spring Data JPA [online]. 2008, 2013 [cit. 2014-05-04]. Dostupné z: http://docs.spring.io/spring-data/jpa/docs/1.4.3.RELEASE/reference/htmlsingle/

[8] PARASCHIV, Eugen. Transactions with Spring 3 and JPA. Baeldung [online]. 2013 [cit. 2014-05-04]. Dostupné z: http://www.baeldung.com/2011/12/26/transaction-configuration-with-jpa-and-spring-3-1/

[9] DENSO WAVE INCORPORATED. QR code [online]. [cit. 2014-05-04]. Dostupné z: http://www.qrcode.com/

[10] PATEL, Viral. Java Virtual Machine, An Inside Story!!. Viralpatel.net [online]. 2008 [cit. 2014-05-04]. Dostupné z: http://viralpatel.net/blogs/java-virtual-machine-an-inside-story/

[11] Security Namespace Configuration. ALEX, Ben a Luke TAYLOR. Spring Security [online]. [cit. 2014-05-04]. Dostupné z: http://docs.spring.io/spring-security/site/docs/3.1.4.RELEASE/reference/ns-config.html

[12] Busy Developers' Guide to HSSF and XSSF Features. The Apache Software Foundation [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://poi.apache.org/spreadsheet/quick-guide.html

[13] Generate Large Excel Report by Using Apache POI Performance Tuning [online]. 2013 [cit. 2014-05-04]. Dostupné z: http://stanicblog.blogspot.sg/2013/07/generate-large-excel-report-by-using.html

[14] SLANINA, Marek. Vytvoření aplikace pro inventarizacina platformě Microsoft Silverlight. Ostrava, 2013. Diplomová práce. Vysoká Škola Báňská - Technická univerzita Ostrava. Vedoucí práce Ing. Jan Skapa, Ph.D.

[15] Novell. Overview: MySQL [online]. 2009 [cit. 2014-05-05]. Dostupné z: http://www.novell.com/documentation/nw65/web_mysql_nw/data/ah7vjv4.html

[16] Data Transfer Object. Martinfowler [online]. 2003 [cit. 2014-05-05]. Dostupné z: http://martinfowler.com/eaaCatalog/dataTransferObject.html

A Obsah přiloženého DVD

Na přiloženém DVD médiu jsou obsaženy tyto položky:

- text diplomové práce v PDF souboru,
- zdrojové kódy aplikace,
- aplikace zabalená do WAR souboru,
- SQL skripty pro serverovou aplikaci
- uživatelská dokumentace
- programátorská dokumentace
- datový slovník